LOLCODE

Chirantan Ekbote

INTRODUCTION

For this project I have implemented an interpreter for LOLCODE, an esoteric programming language inspired by LOLCats. The description of LOLCODE can be found in the Language Implemented section. The interpreter for LOLCODE is written in Python3, taking advantage of the unique features of Python3 to speed up the evaluation process. LOLCODE supports booleans, integers, strings, and floating point numbers. It also supports loops, explicit casting, and recursive functions. The interpreter parses LOLCODE on-the-fly, reading and evaluating each token as it is encountered. In other words, the interpreter does not detect a syntax or semantic error until a runtime call to the offending code block is made. Variables are stored in an Environment - a subclass of Python3's built-in `dict` type, which provides mappings between key-value pairs. Each key or value may be an integer, floating point number, boolean, null, or string. Furthermore, each Environment (except for the global Environment) has an outer Environment, giving functions and loops access to the global scope. Once the interpreter recognizes the tokens it reads in, it performs the Python3 equivalent of the action specified by the given tokens.

DESIGN, PLANNING, AND, EXECUTION

Implementing and testing the interpreter took approximately 50 hours spread out over 6 weeks. I didn't deviate too much from my initial proposal. The few minor changes I did make were made so that it would be easier for me to parse the files and also to give the language more flexibility. For example by requiring that the `MKAY?` keyword terminates every `VISIBLE` statement, programmers can now put expressions and function calls rather than just variables and atoms in the `VISIBLE` statement. Most other changes were similar minor syntactical modifications. The strongest part of my design is the dynamic typing and the way I handle global and local scope. Since the type of a variable is determined at runtime, this gives the programmer a lot of flexibility in how she uses her variables. By subclassing the built-in `dict` type and allowing it to have a parent, the language can very easily handle multiple scopes and keep track of multiple variables with the same name in different scopes. The weakest part of my design is the way loops are currently handled. Given more time, I would have liked to have made the loop functionality more robust. As it is right now, it is nothing more than a simple `while` loop. The easiest art to implement was the parser. I was able to take advantage of Python3's `deque` class and the built-in string operations to easily split up the input into a two-dimensional array of tokens where I could add and remove tokens to either the front or the back of the array. The hardest part to implement was the function handling. I ultimately was able to solve this problem by simply storing the entire code block of a function in a separate `dict` and then appending the appropriate code block to the front of

the code array when a function call was made. This then made it pretty easy to ensure that the interpreter would be able to correctly handle recursive calls as well. I think I did a really good job of implementing my design. The whole interpreter is less than 700 lines of code including comments and properly handles all the features that I wanted it to. I spent about a week making optimizations and adding error checking so that it behaves as close to a real interpreter as possible.

TESTING STRATEGIES

My testing strategy was based around unit testing every individual feature added to the language. Thus each feature has 1-2 average cases and every special case I could come up with. Each test was meant to either show that a particular feature is working correctly or to show that the interpreter gives an error for invalid input. Given more time, I would have liked to test the interpreter on more complex programs that combine several different features. I have a few of this type of programs in my test cases but not as many as I would have liked since I spent most of my time writing up unit tests for individual features. The testing exposed a few logical errors in my code but most of the test cases produced the exact expected output. I have over 120 test cases testing every single feature I wanted to implement and they all work correctly, which has convinced me that I have met the project goals that I set out for myself.

LANGUAGE IMPLEMENTED

This section contains the syntax rules and specifications for LOLCODE.

1. Formatting

    1.Each LOLCODE program must begin with the keyword `HAI` and end with the keyword `KTHXBAI`. Anything before or after these keywords will be ignored.

    2.Whitespace

        1.Spaces will be used to separate tokens in the language, although some tokens contain spaces in them.

        2.Multiple spaces and tabs are treated as a single whitespace token; indentation is irrelevant.

        3.A command starts at the beginning of a line and ends with the newline character. Multiple commands may be placed on the same line if they are separated by a comma (`,`). In this case, the comma acts as a virtual newline or a soft-command-break.

        4.Soft-command-breaks are ignored inside quoted strings. An unterminated string literal will cause an error.

    3.Comments

        1.Single line comments are begun with `BTW` and may occur either on a separate line or after a line of code following a line separator (,).

        The following are all valid single line comments:

```
I HAS A VAR ITZ 12,      BTW VAR = 12
I HAS A VAR ITZ 12
BTW VAR = 12
```

2.Multiline comments are begun with `OBTW` and ended with `TLDR`.

They should be started on their own line, or following a line of code following a line separator. These are valid multiline comments:

```
I HAS A VAR ITZ 12
     OBTW this is a long comment block
          see, i have more comments here
          and here
     TLDR
I HAS A FISH ITZ BOB

I HAS A VAR ITZ 12, OBTW this is a long comment
                    block see, i have more
                    comments here and here
TLDR, I HAS A FISH ITZ BOB
```

2.Variables

1.Scope

1.There is always a global scope.

2.Additionally, each function has its own local scope and any variable defined within the function are defined in this local scope.

3.Functions also have access to the scope of the outer/calling code block. Functions can thus manipulate data outside themselves.

4.A call to a variable defined in more than one scope returns the value of the variable in the innermost scope.

2.Naming

1.Variable identifiers must begin with a letter and must be a combination of letters and numbers. No other symbols are allowed.

2.Variable identifiers are case sensitive. “cheezburger”, “CHEEZBURGER”, and “CheezBurger” would all be different identifiers.

3.Declaration and assignment

1.To declare a variable the keyword `I HAS A` is followed by the variable name. To assign the variable a value within the same statement, follow the variable name with `ITZ <value>`.

2.Assignment of a variable is accomplished with an assignment statement, `<variable> R <expression>`.

3.The following are all valid variable declarations:

```
I HAS A VAR     BTW VAR is null and untyped
VAR R “three”   OBTW VAR is now a YARN and equals
                “three” TLDR
VAR R 3         OBTW VAR is now a NUMBR and equals
                3 TLDR
```

3.Primitive Types

1.The primitive types that LOLCODE recognizes are:

1.strings (YARN)

1.String literals (YARN) are marked with double quotation marks (“).

2.Line continuation *&* soft-command-breaks are ignored inside quoted strings.

3.An unterminated string literal (no closing quote) will cause an error.

2.integers (NUMBR)

1.A NUMBR is an integer as specified in the host implementation/architecture.

2.Any contiguous sequence of digits outside of a quoted YARN and not containing a decimal point (.) is considered a NUMBR.

3.A NUMBR may have a leading hyphen (-) to signify a negative number.

3.floats (NUMBAR)

1.A NUMBAR is a float as specified in the host implementation/architecture.

2.It is represented as a contiguous string of digits containing exactly one decimal point.

3.Casting a NUMBAR to a NUMBR truncates the decimal portion of the floating point number.

4.A NUMBAR may have a leading hyphen (-) to signify a negative number.

4.booleans (TROOF)

1.The two boolean (TROOF) values are WIN (true) and FAIL (false).

2.The empty string ("") and numerical zero are all cast to FAIL. All other values evaluate to WIN.

5.untyped (NOOB)

1.The untyped type (NOOB) cannot be cast into any type except a TROOF. A cast into TROOF makes the variable FAIL.

2.Any operations on a NOOB that assume another type (e.g., math) result in an error.

3.Explicit casts of a NOOB (untyped, uninitialized) to any other type result in an error.

2.Typing is handled dynamically and until a variable is given an initial value, it is untyped (NOOB).

3.IT

1.A bare expression (e.g. a function call or math operation), without any assignment, is a legal statement in LOLCODE.

2.Aside from any side-effects from the expression when evaluated, the final value is placed in the variable IT.

3.IT's value remains in local scope and exists until the next time it is replaced with a bare expression.

4.Operators

1.Calling syntax and precedence

1.Mathematical operators and functions rely on prefix notation. By doing this, it is possible to call and compose operations with a minimum of explicit grouping. When all operators and functions have known arity, no grouping markers are necessary. In cases where operators have variable arity, the operation is closed with `MKAY?`.

2.Calling unary operators then has the following syntax:

```
<operator> <expression>
```

3.The AN keyword can optionally be used to separate arguments, so a binary operator expression has the following syntax:

```
<operator> <expression> AN <expression>
```

4.An expression containing an operator with infinite arity can then be expressed with the following syntax:

```
<operator> <expr1>[[[AN] <expr2>] [AN] <expr3>…] MKAY?
```

2.Math

1.The basic math operations are binary prefix operators:

```
SUM OF <x> AN <y>,         BTW  +
DIFF OF <x> AN <y>,        BTW  -
PRODUKT OF <x> AN <y>,     BTW  *
QUOSHUNT OF <x> AN <y>,    BTW  /
MOD OF <x> AN <y>,         BTW  modulo
BIGGR OF <x> AN <y>,       BTW  max
SMALLR OF <x> AN <y>,      BTW  min
```

2.<x> and <y> are expressions in the above, so mathematical operators can be nested and grouped indefinitely.

3.Math is always performed as floating point math regardless of whether the expressions are NUMBRs or NUMBARs.

4.If one or another of the arguments cannot be safely cast to a numerical type, then the expression fails with an error.

3.Boolean - Boolean operators working on TROOFs are as follows:

```
BOTH OF <x> [AN] <y>,     BTW  and: WIN iff x=WIN, y=WIN
EITHER OF <x> [AN] <y>,   BTW  or: FAIL iff x=FAIL, y=FAIL
NOT <x>,                  BTW unary negation: WIN if x=FAIL
ALL OF <x> [AN] <y> … MKAY?,  BTW infinite arity AND
ANY OF <x> [AN] <y> … MKAY?,  BTW infinite arity OR
```

4.Comparisons

1.Comparison is done with two binary equality operators:

```
BOTH SAEM <x> [AN] <y>,    BTW  WIN iff x == y
DIFFRINT <x> [AN] <y>,    BTW  WIN iff x != y
```

2.Comparisons are performed as floating point math. Otherwise, there is no automatic casting in the equality, so `BOTH SAEM "3" AN 3` is `FAIL`.

3.There are no special numerical comparison operators. Greater-than and similar comparisons are done idiomatically using the minimum and maximum operators.

```
BOTH SAEM <x> AN BIGGR OF <x> AN <y>   BTW x >= y
BOTH SAEM <x> AN SMALLR OF <x> AN <y>  BTW x <= y
DIFFRINT <x> AN SMALLR OF <x> AN <y>   BTW x > y
DIFFRINT <x> AN BIGGR OF <x> AN <y>    BTW x < y
```

4.If <x> in the above formulations is too verbose or difficult to compute, the automatically created IT variable comes in handy. A further idiom could then be:

```
<expression>, DIFFRINT IT AN SMALLR OF IT AN <y>
```

5.Casting

1.Operators that work on specific types implicitly cast parameter values of other types. If the value cannot be safely cast, then it results in an error.

2.An expression's value may be explicitly cast with the binary `MAEK` operator.

```
MAEK <expression> [A] <type>
```

3.Where <type> is one of TROOF, YARN, NUMBR, or NUMBAR. This is only for local casting i.e.,only the resultant value is cast, not the underlying variable(s), if any.

4.To explicitly re-cast a variable, you may create a normal assignment statement with the MAEK operator:

```
<variable> R MAEK <variable> [A] <type>
```

5.Input/Output

1.Input and output in LOLCODE is terminal-based.

2.The print (to STDOUT or the terminal) operator is VISIBLE. It has infinite arity and implicitly concatenates all of its arguments after casting them to YARNs. It is terminated by the keyword MKAY?, followed by the statement delimiter (line end or comma) and the output is automatically terminated with a carriage return.

```
VISIBLE <expression> [<expression> …] MKAY?
```

3.To accept input from the user, the keyword is

```
GIMMEH <variable>
```

which implicitly tries to cast the input as a NUMBR, NUMBAR, TROOF, or NOOB. If the implicit cast fails then the input is assumed to be YARN and the resultant value is stored in the given variable.

6.Flow Control

1.The traditional if-then clause takes advantage of the implicit IT variable and has the form:

```
<expression>
```

```
O RLY?
     YA RLY
          <code block>
     [MEBBE <expression>
          <code block>
     [MEBBE <expression>
          <code block>
     …]]
     [NO WAI
          <code block>]
OIC
```

1.`O RLY?` branches to the block begun with `YA RLY` if `IT` can be cast to `WIN`, and branches to the `NO WAI` block if `IT` is `FAIL`. The code block introduced with `YA RLY` is implicitly closed when `NO WAI` is reached. The `NO WAI` block is closed with `OIC`.

2.Optional `MEBBE <expression>` blocks may appear between the `YA RLY` and `NO WAI` blocks. If the `<expression>` following `MEBBE` is true, then that block is performed; if not, the block is skipped until the following `MEBBE`, `NO WAI`, or `OIC`.

3.An example of this conditional is:

```
BOTH SAEM ANIMAL AN "CAT"
O RLY?
YA RLY, VISIBLE "J00 HAV A KITTEH"
MEBBE BOTH SAEM ANIMAL AN "MAUS"
     VISIBLE "NOM NOM NOM. I EATED IT."
NO WAI, VISIBLE "JOO SUX"
OIC
```

2.Loops

1.Loops are demarcated with `IM IN YR <label>` and `IM OUTTA YR <label>`.

2.Iteration loops have the form:

```
IM IN YR <label> WILE <expression>
     <code block>
IM OUTTA YR <label>
```

3.The `WILE <expression>` evaluates the expression as a `TROOF`: if it evaluates as `WIN`, the loop continues once more, if not, then loop execution stops, and continues after the matching `IM OUTTA YR <label>`.

7.Functions

1.A function is demarcated with the opening keyword `HOW DUZ I` and the closing keyword `IF U SAY SO`. The syntax is as follows:

```
HOW DUZ I <function name> [YR <argument1> [AN YR
<argument2> …]]
     <code_block>
IF U SAY SO
```

2.Currently, the number of arguments in a function can only be defined as a fixed number. The `<argument>`s are single-word identifiers that act as variables within the scope of the function's code. The calling parameters' values are then the initial values for the variables within the function's code block when the function is called.

3.Functions also have access to the global/outer code block’s variables. However if a variable is defined within the function and outside the function, calling the variable returns the value of the variable in the innermost scope.

4.Returning from a function is accomplished as follows:

```
FOUND YR <expression>
```

In the absence of a return statement, the default return value is `NOOB` when the end of the function (`IF U SAY SO`) is reached.

5. A function of given arity is called with:

```
<function name> [<expression1> [<expression2>
[<expression3> …]]] MKAY?
```

BNF GRAMMAR

```
<program>       ::= HAI <nlc> <code_block> <nlc> KTHXBAI

<code_block>    ::= <statement> | <statement> <nlc> <code_block>

<statement>     ::= <loop> | <declaration> | <comment> |
                    <print_block> | <if_block> | <input_block> |
                    <func_decl> | <assignment> | <expression>

<loop>          ::= IM IN YR <label> WILE <expression> <nlc>
                    <code_block> <nlc> IM OUTTA YR <label>

<declaration>   ::= I HAS A <label> | I HAS A <label> ITZ <value>

<comment>       ::= BTW <string> | OBTW <string> TLDR
<print_block>   ::= VISIBLE <expression> <expression>…
                    <expression> MKAY?

<if_block>      ::= O RLY? <nlc> YA RLY <nlc> <code_block> <nlc>
                    OIC | O RLY? <nlc> YA RLY <nlc> <code_block>
                    <nlc> <else_if_block> <nlc> OIC

<else_if_block>::= MEBBE <expression> <nlc> <code_block> <nlc>
                    <else_if_block> | NO WAI <nlc> <code_block> |
                    MEBBE <expression> <nlc> <code_block>

<input_block>   ::= GIMMEH <label>

<func_decl>     ::= HOW DUZ I <label> [[YR <label>] AN YR
                    <label>…] <nlc> <code_block> <nlc> IF U SAY
                    SO

<assignment>    ::= <label> R <expression>

<expression>    ::= <equals> | <both> | <not_equals> | <greater>
                    | <less> | <add> | <sub> | <mul> | <div> |
                    <mod> | <cast> | <either> | <all> | <any> |
                    <not> | <func> | <label> | <atom>

<equals>        ::= BOTH SAEM <expression> AN <expression>

<not_equals>    ::= DIFFRINT <expression> AN <expression>

<both>          ::= BOTH OF <expression> AN <expression>
```

```
<either>        ::= EITHER OF <expression> AN <expression>

<greater>       ::= BIGGR OF <expression> AN <expression>

<less>          ::= SMALLR OF <expression> AN <expression>

<add>           ::= SUM OF <expression> AN <expression>

<sub>           ::= DIFF OF <expression> AN <expression>

<mul>           ::= PRODUKT OF <expression> AN <expression>

<div>           ::= QUOSHUNT OF <expression> AN <expression>

<mod>           ::= MOD OF <expression> AN <expression>

<cast>          ::= MAEK <expression> A <type>

<all>           ::= ALL OF <expression> [AN <expression> …] MKAY?

<any>           ::= ANY OF <expression> [AN <expression> …] MKAY?

<not>           ::= NOT <expression>

<func>          ::= <label> <expression> [<expression> … ] MKAY?

<label>         ::= (A-Za-z)(A-Za-z0-9)*

<atom>          ::= WIN | FAIL | NOOB | (0-9)* | (0-9)*.(0-9)* |
                    "<string>"

<string>        ::= any character string

<nlc>           ::= (newline character) | ,
```

EXAMPLE PROGRAMS

Hello World.lol

```
HAI,           BTW  Hello World
VISIBLE "HAI WORLD!!!!" MKAY?
KTHXBAI
```

Count.lol

```
HAI
I HAS A VAR ITZ 0

BTW for(; VAR <= 10; VAR++)
IM IN YR LOOP WILE BOTH SAEM VAR AN SMALLR OF VAR AN 10
     VISIBLE VAR MKAY?
IM OUTTA YR LOOP
KTHXBAI
```

Fibonacci Numbers.lol

```
HAI,      BTW Fibonacci Numbers

HOW DUZ I FIB YR MAX
    I HAS A NUM ITZ 1
    I HAS A NEXT ITZ 1
    I HAS A COUNT ITZ 0

    DIFFRINT COUNT AN BIGGR OF COUNT AN MAX, BTW if COUNT < MAX
    O RLY?
        YA RLY
            VISIBLE NEXT MKAY?
            COUNT R SUM OF COUNT AN 1,       BTW COUNT++
    OIC

    DIFFRINT COUNT AN BIGGR OF COUNT AN MAX, BTW if COUNT < MAX
    O RLY?
        YA RLY
            VISIBLE NEXT MKAY?
            COUNT R SUM OF COUNT AN 1,  BTW COUNT++
            NEXT R SUM OF NEXT AN NUM,  BTW NEXT = NEXT + NUM
    OIC

    BTW for(; COUNT < MAX; COUNT++)
```

```
    IM IN YR loop WILE DIFFRINT COUNT AN BIGGR OF COUNT AN MAX
        VISIBLE NEXT MKAY?
        SUM OF NEXT AN NUM,         BTW IT = NEXT + NUM
        NUM R NEXT,             BTW NUM = NEXT
        NEXT R IT,              BTW NEXT = IT
        COUNT R SUM OF COUNT AN 1, BTW COUNT += 1
    IM OUTTA YR loop

IF U SAY SO

I HAS A MAX

VISIBLE "How many fibonacci numbers do you wish to see?" MKAY?
GIMMEH MAX

MAX R MAEK MAX A NUMBR

FIB MAX MKAY?

KTHXBAI
```

APPENDIX A:
SOURCE CODE LISTING

```
"""
lolcode.py
Author: Chirantan Ekbote

Simple interpreter for the esoteric programming language LOLCODE.
"""

from collections import deque
import sys, getopt, copy

class Env(dict):
    "An environment: a dict of {'var':val} pairs, with an outer Env."
    def __init__(self, parms=(), args=(), outer=None):
        self.update(zip(parms,args))
        self['IT'] = None
        self.outer = outer
    def find(self, var):
        "Find the innermost Env where var appears."
        return self if var in self else self.outer.find(var)

class FoundException(Exception):
    "A custom exception to catch return statements"
    def __init__(self):
        self.value = 'Found return statement outside function.'
    def __str__(self):
        return self.value

#-----------------------------------------------------------------------------
# Global State Variables
#-----------------------------------------------------------------------------

code = deque([])
global_env = Env()
statement, exp, func = {}, {}, {}
types = { 'TROOF':bool, 'NUMBR':int, 'NUMBAR':float, 'YARN':str }
values = { 'WIN':True, 'FAIL':False, 'NOOB':None }

#-----------------------------------------------------------------------------
# Statement Evaluation Functions
#-----------------------------------------------------------------------------

def found(env=global_env):
    raise FoundException

def funcDecl(env=global_env):
    #print("funcDecl")
    try:
        if (code[0].popleft() != 'HOW' or code[0].popleft() != 'DUZ' or
            code[0].popleft() != 'I'):
            raise IndexError

        name = code[0].popleft()    # Get the function name

        # Get the params
        params = []
        if len(code[0]) > 0:    # We have params
            if code[0].popleft() != 'YR':
                raise IndexError

            params.append(code[0].popleft())
```

```
            while len(code[0]) > 0:
                if code[0].popleft() != 'AN' or code[0].popleft() != 'YR':
                    raise IndexError
                params.append(code[0].popleft())

        code.popleft()  # Pop off function declaration

        # Get the body of the function and add the params to the front
        funcbody = deque([])
        funcbody.append(params)
        while not (len(code[0]) == 4 and code[0][0] == 'IF' and
                   code[0][1] == 'U' and code[0][2] == 'SAY' and
                   code[0][3] == 'SO'):
            funcbody.append(code.popleft())

        # Pop off keywords IF U SAY SO
        funcbody.append(code.popleft())

        # Add to the function dict
        func[name] = funcbody
    except IndexError:
        print("Error: invalid function declaration.")
        sys.exit(1)

def loop(env=global_env):
    #print("loop")
    try:
        if not (code[0].popleft() == 'IM' and code[0].popleft() == 'IN' and
                code[0].popleft() == 'YR'):
            raise IndexError

        # Get the loop name
        name = code[0].popleft()

        # Get the loop condition
        if code[0].popleft() != 'WILE':
            raise IndexError
        cond = code.popleft()

        # Get the body of the loop
        codeloop = deque([])
        while not (len(code[0]) == 4 and code[0][0] == 'IM' and
                   code[0][1] == 'OUTTA' and code[0][2] == 'YR' and
                   code[0][3] == name):
            codeloop.append(code.popleft())

        # Evaluate loop
        code.appendleft(deque(cond))
        while(expression(env)):
            # Add the code for the body of the loop and run it
            code.extendleft(reversed(copy.deepcopy(codeloop)))
            while not (len(code[0]) == 4 and code[0][0] == 'IM' and
                       code[0][1] == 'OUTTA' and code[0][2] == 'YR'
                       and code[0][3] == name):
                codeBlock(env)

            # Put the loop condition expression back in
            code.appendleft(deque(cond))

        # Remove end of loop keywords
        code.popleft()
```

```
    except IndexError:
        print("Error: invalid loop.")
        sys.exit(1)

def inputBlock(env=global_env):
    #print("inputBlock")
    val = input('LOL>> ')
    try:
        scope = env.find(code[0][1])
    except AttributeError:  # Variable is not defined
        print("Error: invalid GIMMEH statement.")
        sys.exit(1)
    try:
        scope[code[0][1]] = values[val]
    except KeyError:    # Not WIN, FAIL, or NOOB
        try:
            scope[code[0][1]] = int(val)
        except ValueError:  # Not an int
            try:
                scope[code[0][1]] = float(val)
            except ValueError:
                # We have a string
                scope[code[0][1]] = val
    code.popleft()  # Pop off GIMMEH statement

def printBlock(env=global_env):
    #print("printBlock")
    try:
        code[0].popleft()
        out = ''
        while code[0][0] != 'MKAY?':
            out += str(expression(env)).strip('"') + ' '
        print(out)
        code.popleft()
    except (KeyError, IndexError):
        print("Error: invalid VISIBLE statement.")
        sys.exit(1)

def assignment(env=global_env):
    #print("assignment")
    try:
        name = code[0].popleft()        # pop off variable name

        if code[0].popleft() != 'R':    # Pop off keyword R
            raise IndexError

        env.find(name)[name] = expression(env)
    except AttributeError:  # The variable has not been previously defined
        env[name] = expression(env)
    except IndexError:
        print("Error: invalid assignment.")
        sys.exit(1)

def declaration(env=global_env):
    #print("declaration")
    try:
        if (code[0].popleft() != 'I' or code[0].popleft() != 'HAS' or
            code[0].popleft() != 'A'):
            raise IndexError

        name = code[0].popleft()
```

```
        if name in env: # Variable has been previously defined in this scope
            raise IndexError

        if len(code[0]) > 0 and code[0].popleft() == 'ITZ':
            env[name] = expression(env)
        else:
            env[name] = None
            code.popleft()
    except IndexError:
        print("Error: Invalid declaration.")
        sys.exit(1)

def elseIfBlock(env=global_env, depth=0):
    while code[0][0] != 'MEBBE' and code[0][0] != 'NO' and code[0][0] != 'OIC':
        code.popleft()

    if code[0][0] == 'MEBBE':
        code[0].popleft()
        # Check the MEBBE expression and evaluate if true
        if expression(env) == True:
            while (code[0][0] != 'MEBBE' and code[0][0] != 'NO' and
                   code[0][0] != 'OIC'):
                codeBlock(env)
        else:
            elseIfBlock(env, depth + 1)    # This MEBBE was not true
    elif code[0][0] == 'NO' and code[0][1] == 'WAI':
        code.popleft()
        # Evaluate the NO WAI code block
        while (code[0][0] != 'MEBBE' and code[0][0] != 'NO' and
               code[0][0] != 'OIC'):
            codeBlock(env)

    # Make sure we've reached OIC, end of if block
    while depth == 0 and code[0][0] != 'OIC':
        code.popleft()
    code.popleft()  # Pop off keyword OIC

def ifBlock(env=global_env):
    #print("ifblock")
    try:
        if code[0][0] != 'O' or code[0][1] != 'RLY?':
            raise IndexError
        # Pop off keywords and evaluate if block
        code.popleft()
        if env['IT']:
            # Find the YA RLY keyword
            while 'YA' not in code[0] and 'RLY' not in code[0]:
                code.popleft()

            # Pop off YA RLY
            code.popleft()

            # Execute statements
            while (code[0][0] != 'MEBBE' and code[0][0] != 'NO' and
                   code[0][0] != 'OIC'):
                codeBlock(env)

            # Make sure we've reached keyword OIC
            while code[0][0] != 'OIC':
                code.popleft()
```

```
            # Now pop that off
            code.popleft()
        else:
            elseIfBlock(env)
    except IndexError:
        print("Error: invalid O RLY? block.")
        sys.exit(1)

def comment(env=global_env):
    #print("comment")
    if code[0][0] == 'BTW':
        code.popleft()
    elif code[0][0] == 'OBTW':
        try:
            while code[0][-1] != 'TLDR':    # Last token is not TLDR
                code.popleft()
            code.popleft()                  # Pop off line with TLDR
        except IndexError as err:
            print("Error: unexpected end of file. Expecting keyword TLDR")
            sys.exit(1)

#-----------------------------------------------------------------------------
# Binary Operators and Functions
#-----------------------------------------------------------------------------

def funcEval(env=global_env):
    #print("functionEval")
    # get the body of the function
    funcbody = copy.deepcopy(func[code[0].popleft()])

    # Get the params and create the environment
    params = funcbody.popleft()
    args = []
    while code[0][0] != 'MKAY?':
        args.append(expression(env))

    # Make sure we have the coorect number of arguments
    if len(args) != len(params):
        print("Error: invalid number of arguments in function call.")
        sys.exit(1)

    code[0].popleft()  # pop off MKAY?
    funcEnv = Env(params, args, env)    # Create the function environment

    # Add the function code and evaluate
    out = None
    code.extendleft(reversed(funcbody))
    try:
        while not (len(code[0]) == 4 and code[0][0] == 'IF' and
                   code[0][1] == 'U' and code[0][2] == 'SAY' and
                   code[0][3] == 'SO'):
            codeBlock(funcEnv)

        # We've reached the end so pop off the last keywords
        code.popleft()
    except FoundException:
        # We've reached a return statement
        if code[0].popleft() != 'FOUND' or code[0].popleft() != 'YR':
            print('Error: invalid FOUND statement.')
            sys.exit(1)
        out = expression(funcEnv)
```

```
        # Now get rid of excess function code
        while not (len(code[0]) == 4 and code[0][0] == 'IF' and
                   code[0][1] == 'U' and code[0][2] == 'SAY' and
                   code[0][3] == 'SO'):
            code.popleft()

        # pop off keywords
        code.popleft()
    finally:
        # Delete function environment and return output
        del funcEnv
        return out

def getBinArgs(env=global_env):
    # Get first expression
    x = expression(env)

    # Check for separator keyword AN
    if (code[0].popleft() != 'AN'):
        raise IndexError

    # Get second expression
    y = expression(env)

    return (x, y)

def both(env=global_env):
    try:
        if (code[0].popleft() != 'BOTH' or code[0].popleft() != 'OF'):
            raise IndexError

        # Get arguments
        x, y = getBinArgs(env)

        return bool(x and y)
    except IndexError:
        print("Error: invalid BOTH comparison.")
        sys.exit(1)


def either(env=global_env):
    try:
        if (code[0].popleft() != 'EITHER' or code[0].popleft() != 'OF'):
            raise IndexError

        # Get arguments
        x, y = getBinArgs(env)

        return bool(x or y)
    except IndexError:
        print("Error: invalid EITHER comparison.")
        sys.exit(1)


def neg(env=global_env):
    try:
        if code[0].popleft() != 'NOT':
            raise IndexError

        # Get expression value
```

```
        x = expression(env)

        return not x
    except IndexError:
        print("Error: invalid NOT operation.")
        sys.exit(1)


def allOf(env=global_env):
    try:
        if (code[0].popleft() != 'ALL' or code[0].popleft() != 'OF'):
            raise IndexError

        # Get arguments
        args = []
        while code[0][0] != 'MKAY?':
            args.append(expression(env))
            # See if there are any more arguments
            if code[0][0] == 'AN':
                code[0].popleft()   # Pop it off
        code[0].popleft()   # Pop off MKAY?

        return all(args)
    except IndexError:
        print("Error: invalid ALL operation.")
        sys.exit(1)

def anyOf(env=global_env):
    try:
        if (code[0].popleft() != 'ANY' or code[0].popleft() != 'OF'):
            raise IndexError

        # Get arguments
        args = []
        while code[0][0] != 'MKAY?':
            args.append(expression(env))
            # See if there are any more arguments
            if code[0][0] == 'AN':
                code[0].popleft()   # Pop it off
        code[0].popleft()  # pop off MKAY?

        return any(args)
    except IndexError:
        print("Error: invalid ANY operation.")
        sys.exit(1)

def cast(env=global_env):
    #print("cast")
    try:
        code[0].popleft()       # Pop off keyword MAEK
        exp = expression(env)

        if code[0].popleft() != 'A':
            raise IndexError

        return types[code[0].popleft()](exp)
    except (AttributeError, KeyError, IndexError, TypeError):
        print("Error: invalid cast.")
        sys.exit(1)

#------------------------------------------------------------------------
```

```
# Math Operations and Comparisons
#-------------------------------------------------------------------------

def equals(env=global_env):
    #print('equals')
    try:
        if (code[0].popleft() != 'BOTH' or code[0].popleft() != 'SAEM'):
            raise IndexError

        # Get arguments
        x, y = getBinArgs(env)

        return x == y
    except IndexError:
        print("Error: invalid equals comparison.")
        sys.exit(1)

def notEquals(env=global_env):
    try:
        if (code[0].popleft() != 'DIFFRINT'):
            raise IndexError

        # Get arguments
        x, y = getBinArgs(env)

        return (x != y)
    except IndexError:
        print("Error: invalid not equals comparison.")
        sys.exit(1)

def greater(env=global_env):
    try:
        if (code[0].popleft() != 'BIGGR' or code[0].popleft() != 'OF'):
            raise IndexError

        # Get arguments
        x, y = getBinArgs(env)

        return max(x, y)
    except IndexError:
        print("Error: invalid greater than comparison.")
        sys.exit(1)

def less(env=global_env):
    try:
        if (code[0].popleft() != 'SMALLR' or code[0].popleft() != 'OF'):
            raise IndexError

        # Get arguments
        x, y = getBinArgs(env)

        return min(x, y)
    except IndexError:
        print("Error: invalid less than comparison.")
        sys.exit(1)

def add(env=global_env):
    try:
        if (code[0].popleft() != 'SUM' or code[0].popleft() != 'OF'):
            raise IndexError
```

```
        # Get arguments
        x, y = getBinArgs(env)

        return x + y
    except IndexError:
        print("Error: invalid addition operation.")
        sys.exit(1)

def sub(env=global_env):
    try:
        if (code[0].popleft() != 'DIFF' or code[0].popleft() != 'OF'):
            raise IndexError

        # Get arguments
        x, y = getBinArgs(env)

        return x - y
    except IndexError:
        print("Error: invalid subtraction operation.")
        sys.exit(1)

def mul(env=global_env):
    try:
        if (code[0].popleft() != 'PRODUKT' or code[0].popleft() != 'OF'):
            raise IndexError

        # Get arguments
        x, y = getBinArgs(env)

        return x * y
    except IndexError:
        print("Error: invalid multiplication operation.")
        sys.exit(1)

def div(env=global_env):
    try:
        if (code[0].popleft() != 'QUOSHUNT' or code[0].popleft() != 'OF'):
            raise IndexError

        # Get arguments
        x, y = getBinArgs(env)

        return x / y
    except IndexError:
        print("Error: invalid division operation.")
        sys.exit(1)

def mod(env=global_env):
    try:
        if (code[0].popleft() != 'MOD' or code[0].popleft() != 'OF'):
            raise IndexError

        # Get arguments
        x, y = getBinArgs(env)

        return x % y
    except IndexError:
        print("Error: invalid modulo operation.")
        sys.exit(1)

def string(env=global_env):
```

```
    try:
        if '"' not in code[0][0]:
            raise IndexError

        out = code[0].popleft()
        # Test to see if we have a long string
        if out.count('"') < 2:
            while '"' not in code[0][0]:
                out += ' ' + code[0].popleft()
            out += ' '+ code[0].popleft()

        return out.strip('"')
    except IndexError:
        print("Error: invalid YARN.")
        sys.exit(1)

def atom(env=global_env):
    #print("atom")
    try:
        out = values[code[0][0]]
        code[0].popleft()
    except KeyError:
        try:
            out = int(code[0][0])
            code[0].popleft()
        except ValueError:
            try:
                out = float(code[0][0])
                code[0].popleft()
            except ValueError:
                # We have a string
                out = string()
    return out

def bothBranch(env=global_env):
    try:
        if code[0][1] == 'OF':
            return both(env)
        else:
            return equals(env)
    except IndexError:
        print("Error: invalid BOTH expression.")
        sys.exit(1)

#---------------------------------------------------------------------------
# Main Expression Evaluation Function
#---------------------------------------------------------------------------

# Create expression dispatch dictionary
exp = { 'BOTH':bothBranch, 'DIFFRINT':notEquals, 'BIGGR':greater, 'SMALLR':less,
        'SUM':add, 'DIFF':sub, 'PRODUKT':mul, 'QUOSHUNT':div, 'MOD':mod,
        'MAEK':cast, 'EITHER':either, 'ALL':allOf, 'ANY':anyOf, 'NOT':neg }

def expression(env=global_env):
    #print("expression")
    out = None
    try:
        out = exp[code[0][0]](env)
    except KeyError:
        try:
            out = env.find(code[0][0])[code[0][0]]
```

```
            #print("variable")
            code[0].popleft()
        except AttributeError:  # code[0][0] is not a variable
            if code[0][0] in func:
                #print('function')
                out = funcEval(env)
            else:
                out = atom(env)

    # If there's nothing left, pop off empty command list
    if len(code[0]) == 0:
        code.popleft()

    return out

#-----------------------------------------------------------------------
# Main Program
#-----------------------------------------------------------------------

# Create statement dispatch dictionary
statement = {'IM':loop, 'I':declaration, 'BTW':comment, 'VISIBLE':printBlock,
             'O':ifBlock, 'OBTW':comment, 'FOUND':found, 'GIMMEH':inputBlock,
             'HOW':funcDecl}

def codeBlock(env=global_env):
    try:
        statement[code[0][0]](env)
    except KeyError:
        if 'R' in code[0]:  # Maybe it's an assignment
            assignment(env)
        else:               # Must be an expression
            env['IT'] = expression(env)

def program():
    try:
        while code[0][0] != 'HAI':
            code.popleft()

        # Now the program starts
        code.popleft()
        while code[0][0] != 'KTHXBAI':
            codeBlock(global_env)
        sys.exit()
    except IndexError:
        print("Error: unexpected end of file. Expecting keyword KTHXBAI")
        sys.exit(1)
    except Exception as err:
        print("Unexpected error:", err)
        sys.exit(1)

if __name__ == "__main__":
    try:
        optlist, args = getopt.getopt(sys.argv[1:], 'f:')
    except getopt.GetoptError:
        print(err)
        sys.exit(1)

    file = None
    for option, value in optlist:
        if option == '-f':
            file = open(value, 'r')
```

```
for line in file:
    if len(line) > 1: #There is more than just a newline character
        softbreaks = line.split(',')
        for token in softbreaks:
            code.append(deque(token.split()))

program()
```

APPENDIX B:
TEST CASES

1-STRUCTURE

This test checks that commas separate multiple statements on the same line.

```
HAI
        VISIBLE "Lorem" MKAY?, VISIBLE "ipsum" MKAY?, VISIBLE "dolor"
MKAY?, VISIBLE "sit" MKAY?
KTHXBAI
```

Output:

```
Lorem
ipsum
dolor
sit
```

```
This test checks to see whether the bare minimum program--consisting only
of a main block--is correctly interpreted and produces no output.
HAI
KTHXBAI

Output:
```

This test checks to see whether line indentation is ignored.

```
        HAI
        VISIBLE "Lorem ipsum dolor sit" MKAY?
        KTHXBAI
```

Output:

```
Lorem ipsum dolor sit
```

```
This test makes sure a program begins with the HAI keyword.
KTHXBAI

Output:
Error: unexpected end of file. Expecting keyword KTHXBAI
```

```
This test makes sure a program ending without a KTHXBYE statement generates
an error.
HAI

Output:
Error: unexpected end of file. Expecting keyword KTHXBAI
```

This test checks to make sure the carriage return character (\r) indicates the end of a statement.

```
HAI
        VISIBLE "Lorem" MKAY?
        VISIBLE "ipsum" MKAY?
        VISIBLE "dolor" MKAY?
        VISIBLE "sit" MKAY?
KTHXBAI
```

Output:

```
Lorem
ipsum
dolor
sit
```

This test checks to make sure the carriage return and linefeed characters (\r\n) indicate the end of a statement.

```
HAI
        VISIBLE "Lorem" MKAY?
        VISIBLE "ipsum" MKAY?
        VISIBLE "dolor" MKAY?
        VISIBLE "sit" MKAY?
KTHXBAI
```

Output:

```
Lorem
ipsum
dolor
sit
```

This test checks to make sure the linefeed character (\n) indicates the end of a statement.

```
HAI
        VISIBLE "Lorem" MKAY?
        VISIBLE "ipsum" MKAY?
        VISIBLE "dolor" MKAY?
        VISIBLE "sit" MKAY?
KTHXBAI
```

Output:

```
Lorem
ipsum
dolor
sit
```

This test checks that whitespace in between tokens is handled properly. It tests whitespace only, tabs only, alternating whitespace and tabs, and alternating tabs and whitespace.

```
HAI
        VISIBLE    "Lorem "    "ipsum "    "dolor "    "sit" MKAY?
        VISIBLE                 "Lorem "                        "ipsum "
                        "dolor "                        "sit" MKAY?
        VISIBLE          "Lorem "        "ipsum "        "dolor "
"sit" MKAY?
        VISIBLE         "Lorem "                "ipsum "
        "dolor "                "sit" MKAY?
KTHXBAI
```

Output:

```
Lorem  ipsum  dolor  sit
Lorem  ipsum  dolor  sit
Lorem  ipsum  dolor  sit
Lorem  ipsum  dolor  sit
```

2-COMMENTS

1-Single Line

This test checks that comments may appear after the end of the main block.

```
HAI
        VISIBLE "Lorem ipsum dolor sit" MKAY?
KTHXBAI
BTW Lorem ipsum dolor sit
```

Output:

```
Lorem ipsum dolor sit
```

This test checks that comments may appear before the start of the main block.

```
BTW Lorem ipsum dolor sit
HAI
        VISIBLE "Lorem ipsum dolor sit" MKAY?
KTHXBAI
```

Output:

```
Lorem ipsum dolor sit
```

This test checks that comments may appear at the end of a line.

```
HAI 1.2
        VISIBLE "Lorem ipsum dolor sit" MKAY?, BTW Lorem ipsum dolor sit
KTHXBAI
```

Output:

```
Lorem ipsum dolor sit
```

This test checks that comments may appear on their own line.

```
HAI
        VISIBLE "Lorem ipsum dolor sit" MKAY?
        BTW Lorem ipsum dolor sit
KTHXBAI
```

Output:

```
Lorem ipsum dolor sit
```

2-COMMENTS

2-Multiple Line

This test checks that multiple line comments may appear after the end of the main block.

```
HAI
        VISIBLE "Lorem ipsum dolor sit" MKAY?
KTHXBAI
OBTW
Lorem ipsum dolor sit amet, consectetur adipiscing elit.  Quisque
hendrerit, arcu ut porttitor congue, odio sapien mattis mauris, eget semper
nisi quam non enim. Integer aliquam, lacus non euismod aliquet, dolor lorem
rhoncus erat, sed iaculis quam erat varius lacus. Aenean ullamcorper vehicula
leo ut imperdiet.  Vestibulum egestas nisl in dui laoreet ultrices.  Vestibulum
at commodo mi. Sed at eros mauris, sed pellentesque augue. Ut a dolor metus,
eget hendrerit enim.  Nulla eros nibh, placerat sit amet eleifend id, lobortis
ut felis. Donec tempus pulvinar magna at euismod. Aenean tempor lacus a velit
tincidunt eu fringilla ligula adipiscing. Suspendisse leo dui, gravida vel
fringilla non, consectetur id mi.
TLDR
```

Output:

```
Lorem ipsum dolor sit
```

This test checks that multiple line comments may appear before the start of the main block.

```
OBTW
Lorem ipsum dolor sit amet, consectetur adipiscing elit.  Quisque
hendrerit, arcu ut porttitor congue, odio sapien mattis mauris, eget semper
nisi quam non enim. Integer aliquam, lacus non euismod aliquet, dolor lorem
rhoncus erat, sed iaculis quam erat varius lacus. Aenean ullamcorper vehicula
leo ut imperdiet.  Vestibulum egestas nisl in dui laoreet ultrices.  Vestibulum
at commodo mi. Sed at eros mauris, sed pellentesque augue. Ut a dolor metus, eget
hendrerit enim.  Nulla eros nibh, placerat sit amet eleifend id, lobortis
ut felis. Donec tempus pulvinar magna at euismod. Aenean tempor lacus a velit
tincidunt eu fringilla ligula adipiscing. Suspendisse leo dui, gravida vel
fringilla non, consectetur id mi.
TLDR
HAI
        VISIBLE "Lorem ipsum dolor sit" MKAY?
KTHXBAI
```

Output:

```
Lorem ipsum dolor sit
```

This test checks that multiple line comments may begin after a line separator.

```
HAI
        VISIBLE "Lorem ipsum" MKAY?, OBTW Lorem ipsum dolor sit amet, consectetur
adipiscing elit.  Quisque hendrerit, arcu ut porttitor congue, odio sapien mattis
mauris, eget semper nisi quam non enim. Integer aliquam, lacus non euismod aliquet,
dolor lorem rhoncus erat, sed iaculis quam erat varius lacus.
Aenean ullamcorper vehicula leo ut imperdiet.  Vestibulum egestas nisl in dui
laoreet ultrices.  Vestibulum at commodo mi. Sed at eros mauris, sed
pellentesque augue. Ut a dolor metus, eget hendrerit enim.  Nulla eros nibh,
placerat sit amet eleifend id, lobortis ut felis. Donec tempus pulvinar magna at
euismod. Aenean tempor lacus a velit tincidunt eu fringilla ligula
adipiscing. Suspendisse leo dui, gravida vel fringilla non, consectetur id mi.
TLDR
        VISIBLE "dolor sit" MKAY?
KTHXBAI
```

Output:

```
Lorem ipsum
dolor sit
```

This test checks that multiple line comments must end on their own line and not be followed by any other code.

```
HAI
        VISIBLE "Lorem ipsum" MKAY?
        OBTW
Lorem ipsum dolor sit amet, consectetur adipiscing elit.  Quisque hendrerit, arcu ut
porttitor congue, odio sapien mattis mauris, eget semper nisi quam non
enim. Integer aliquam, lacus non euismod aliquet, dolor lorem rhoncus erat, sed
iaculis quam erat varius lacus. Aenean ullamcorper vehicula leo ut imperdiet.
Vestibulum egestas nisl in dui laoreet ultrices.  Vestibulum at commodo mi. Sed
at eros mauris, sed pellentesque augue. Ut a dolor metus, eget hendrerit enim.
Nulla eros nibh, placerat sit amet eleifend id, lobortis ut felis. Donec tempus
pulvinar magna at euismod. Aenean tempor lacus a velit tincidunt eu fringilla
ligula adipiscing. Suspendisse leo dui, gravida vel fringilla non, consectetur
id mi. TLDR VISIBLE "dolor sit" MKAY?
KTHXBAI
```

Output:

```
Lorem ipsum
Error: unexpected end of file. Expecting keyword TLDR
```

This test checks that multiple line comments must start on a separate line from other code.

```
HAI
        VISIBLE "Lorem ipsum" MKAY? OBTW
Lorem ipsum dolor sit amet, consectetur adipiscing elit.  Quisque hendrerit, arcu
ut porttitor congue, odio sapien mattis mauris, eget semper nisi quam non
enim. Integer aliquam, lacus non euismod aliquet, dolor lorem rhoncus erat, sed
iaculis quam erat varius lacus. Aenean ullamcorper vehicula leo ut imperdiet.
Vestibulum egestas nisl in dui laoreet ultrices.  Vestibulum at commodo mi. Sed
at eros mauris, sed pellentesque augue. Ut a dolor metus, eget hendrerit enim.
Nulla eros nibh, placerat sit amet eleifend id, lobortis ut felis. Donec tempus
pulvinar magna at euismod. Aenean tempor lacus a velit tincidunt eu fringilla
ligula adipiscing. Suspendisse leo dui, gravida vel fringilla non, consectetur
id mi. TLDR
        VISIBLE "dolor sit" MKAY?
KTHXBAI
```

Output:

```
Lorem ipsum
Error: invalid YARN.
```

This test checks that multiple line comments may end on a separate line and start on the same line as the first line of the comment.

```
HAI
        VISIBLE "Lorem ipsum" MKAY?
        OBTW Lorem ipsum dolor sit amet, consectetur adipiscing elit.  Quisque
hendrerit, arcu ut porttitor congue, odio sapien mattis mauris, eget semper
nisi quam non enim. Integer aliquam, lacus non euismod aliquet, dolor lorem
rhoncus erat, sed iaculis quam erat varius lacus. Aenean ullamcorper vehicula
leo ut imperdiet.  Vestibulum egestas nisl in dui laoreet ultrices.  Vestibulum
at commodo mi. Sed at eros mauris, sed pellentesque augue. Ut a dolor metus, eget
hendrerit enim.  Nulla eros nibh, placerat sit amet eleifend id, lobortis
ut felis. Donec tempus pulvinar magna at euismod. Aenean tempor lacus a velit
tincidunt eu fringilla ligula adipiscing. Suspendisse leo dui, gravida vel
fringilla non, consectetur id mi.
        TLDR
        VISIBLE "dolor sit" MKAY?
KTHXBAI
```

Output:

```
Lorem ipsum
dolor sit
```

This test checks that multiple line comments may start and end on separate lines.

```
HAI
        VISIBLE "Lorem ipsum" MKAY?
        OBTW
Lorem ipsum dolor sit amet, consectetur adipiscing elit.  Quisque hendrerit, arcu
ut porttitor congue, odio sapien mattis mauris, eget semper nisi quam non
enim. Integer aliquam, lacus non euismod aliquet, dolor lorem rhoncus erat, sed
iaculis quam erat varius lacus. Aenean ullamcorper vehicula leo ut imperdiet.
Vestibulum egestas nisl in dui laoreet ultrices.  Vestibulum at commodo mi. Sed
at eros mauris, sed pellentesque augue. Ut a dolor metus, eget hendrerit enim.
Nulla eros nibh, placerat sit amet eleifend id, lobortis ut felis. Donec tempus
pulvinar magna at euismod. Aenean tempor lacus a velit tincidunt eu fringilla
ligula adipiscing. Suspendisse leo dui, gravida vel fringilla non, consectetur
id mi.
        TLDR
        VISIBLE "dolor sit" MKAY?
KTHXBAI
```

Output:

```
Lorem ipsum
dolor sit
```

This test checks that multiple line comments may start on a separate line and end on the same line as the last line of the comment.

```
HAI
        VISIBLE "Lorem ipsum" MKAY?
        OBTW
Lorem ipsum dolor sit amet, consectetur adipiscing elit.  Quisque hendrerit, arcu
ut porttitor congue, odio sapien mattis mauris, eget semper nisi quam non
enim. Integer aliquam, lacus non euismod aliquet, dolor lorem rhoncus erat, sed
iaculis quam erat varius lacus. Aenean ullamcorper vehicula leo ut imperdiet.
Vestibulum egestas nisl in dui laoreet ultrices.  Vestibulum at commodo mi. Sed
at eros mauris, sed pellentesque augue. Ut a dolor metus, eget hendrerit enim.
Nulla eros nibh, placerat sit amet eleifend id, lobortis ut felis. Donec tempus
pulvinar magna at euismod. Aenean tempor lacus a velit tincidunt eu fringilla
ligula adipiscing. Suspendisse leo dui, gravida vel fringilla non, consectetur
id mi. TLDR
        VISIBLE "dolor sit" MKAY?
KTHXBAI
```

Output:

```
Lorem ipsum
dolor sit
```

3-TYPES

1-Nil

This test makes sure a nil typed value can be cast to a boolean typed value.

```
HAI
        I HAS A var, BTW var is NOOB
        BTW Implicitly cast NOOB to TROOF and then TROOF to NUMBR and
finally
        BTW NUMBR to YARN to print.
        VISIBLE SUM OF 0 AN NOT var MKAY?
KTHXBAI
```

Output:

```
1
```

This test makes sure a nil typed value can NOT be cast to a floating point decimal typed value.

```
HAI
        I HAS A var, BTW var is NOOB
        SUM OF 0.0 AN var
KTHXBAI
```

Output:

```
Unexpected error: unsupported operand type(s) for +: 'float' and 'NoneType'
```

This test makes sure a nil typed value can NOT be cast to an integer typed value.

```
HAI
        I HAS A var, BTW var is NOOB
        SUM OF 0 AN var
KTHXBAI
```

```
Unexpected error: unsupported operand type(s) for +: 'int' and 'NoneType'
```

This test makes sure a nil typed value can be cast to a string typed value.

```
HAI
        I HAS A var, BTW var is NOOB
        VISIBLE var MKAY?
KTHXBAI
```

Output:
```
None
```

3-TYPES

2-Boolean

This test checks how different values of each type are cast to boolean types.

```
HAI
        I HAS A var1
        I HAS A var2 ITZ ""
        I HAS A var3 ITZ 0
        I HAS A var4 ITZ 0.00
        I HAS A var5 ITZ "Lorem ipsum dolor sit"
        I HAS A var6 ITZ 1
        I HAS A var7 ITZ 2.345
        VISIBLE SUM OF 0 AN NOT var1 MKAY?
        VISIBLE SUM OF 0 AN NOT var2 MKAY?
        VISIBLE SUM OF 0 AN NOT var3 MKAY?
        VISIBLE SUM OF 0 AN NOT var4 MKAY?
        VISIBLE SUM OF 0 AN NOT var5 MKAY?
        VISIBLE SUM OF 0 AN NOT var6 MKAY?
        VISIBLE SUM OF 0 AN NOT var7 MKAY?
KTHXBAI
```

Output:

```
1
0
1
1
0
0
0
```

3-TYPES

3-Integer

```
This test checks that dividing by zero gives an error.
HAI
        I HAS A var ITZ -123
        VISIBLE QUOSHUNT OF var AN 0 MKAY?
KTHXBAI

Output:
Unexpected error: division by zero
```

This test checks that other types are implicitly cast to integers
correctly.

```
HAI
        I HAS A var1 ITZ FAIL
        I HAS A var2 ITZ WIN
        I HAS A var3 ITZ "123"
        VISIBLE SUM OF 0 AN var1 MKAY?
        VISIBLE SUM OF 0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN var3 MKAY?
KTHXBAI
```

Output:

```
0
1
Unexpected error: unsupported operand type(s) for +: 'int' and 'str'
```

This test checks that the hyphen which signifies negatively valued integers appears immediately before the number.

```
HAI
        I HAS A var ITZ - 123
        MAEK var a NUMBR
KTHXBAI
```

Output:

```
Error: invalid YARN.
```

This test checks that negatively valued integers are parsed and stored correctly.

```
HAI
        I HAS A var ITZ -123
        VISIBLE var MKAY?
KTHXBAI
```

Output:

```
-123
```

3-TYPES

4-Float

This test checks to make sure that negative floating point values are signified by an immediately adjacent hyphen and raise an error if not.

```
HAI
        I HAS A var ITZ - 1.23
        VISIBLE SUM OF 1.0 AN var MKAY?
KTHXBAI
```

Output:
```
Error: invalid YARN.
```

This test checks to make sure only one decimal point is allowed in floating point decimal numbers.

```
HAI
        I HAS A var ITZ 1..23
        VISIBLE SUM OF 1.0 AN var MKAY?
KTHXBAI
```

Output:

```
Error: invalid YARN.
```

3-TYPES

5-String

This test checks that strings which contain embedded comment keywords work correctly.

```
HAI
        VISIBLE "Lorem ipsum BTW dolor sit" MKAY?
        VISIBLE "Lorem ipsum OBTW dolor TLDR sit" MKAY?
KTHXBAI
```

Output:

```
Lorem ipsum BTW dolor sit
Lorem ipsum OBTW dolor TLDR sit
```

This test checks that strings which do not have a closing quote result in an error.

```
HAI
        VISIBLE "Lorem ipsum dolor sit MKAY?
KTHXBAI
```

Output:
```
Error: invalid YARN.
```

4-OUTPUT

This test makes sure the print expression prints basic string input (i.e. no string character excape sequences) correctly.

```
HAI
        VISIBLE "Lorem " "ipsum " "dolor " "sit" MKAY?
KTHXBAI
```

Output:

```
Lorem  ipsum  dolor  sit
```

This test makes sure the print expression print True and False for the WIN and FAIL keywords.

```
HAI
        VISIBLE WIN FAIL MKAY?
KTHXBAI
```

Output:

```
True False
```

This test makes sure the print expression prints floating point decimals correctly.

```
HAI
        VISIBLE 1.00 2.345 6.789 MKAY?
KTHXBAI
```

Output:

```
1.0 2.345 6.789
```

This test makes sure the print expression prints integers correctly.

```
HAI
        VISIBLE 1 2 3 4 5 6 7 8 9 10 MKAY?
        VISIBLE -23 4 5 MKAY?
KTHXBAI
```

Output:

```
1 2 3 4 5 6 7 8 9 10
-23 4 5
```

This test makes sure the print expression works with various types as arguments.

```
HAI
        VISIBLE 1 2.345 "Lorem ipsum dolor sit" MKAY?
KTHXBAI
```

Output:

```
1 2.345 Lorem ipsum dolor sit
```

This test makes sure the print expression prints an empty line if no args are present.

```
HAI
        VISIBLE MKAY?
KTHXBAI
```

Output:

5-INPUT

This test checks to see whether a program correctly accepts input via the GIMMEH statement. It keeps asking for input until the user enters QUIT.

```
HAI
        I HAS A var
        GIMMEH var
        IM IN YR loop WILE DIFFRINT var AN "QUIT"
                VISIBLE var MKAY?
                GIMMEH var
        IM OUTTA YR loop
KTHXBAI
```

Output:

```
LOL>> 2
LOL>> 3
LOL>> test
LOL>> hello
LOL>> this
LOL>> is
LOL>> output
LOL>> QUIT
```

6-VARIABLES

This test checks that variable assignment works correctly.

```
HAI
        I HAS A var1
        I HAS A var2
        I HAS A var3
        var1 R 1
        var2 R 2.345
        var3 R "Lorem ipsum dolor sit"
        VISIBLE var1 var2 var3 MKAY?
KTHXBAI
```

Output:

```
1 2.345 Lorem ipsum dolor sit
```

```
OBTW Since TEST is not declared in this function, if it has been declared
in one of the parents of this function, then that variable's value will be
changed. In other words, this shows how functions have access to global
variables. This has been purposefully left in to give the programmer more
power when it comes to manipulating variables. However if in doubt, the
programmer should always declare variable before using them. TLDR

HAI,    BTW testing assignment statements

HOW DUZ I testAssign YR value
    TEST R value
    FOUND YR TEST
IF U SAY SO

HOW DUZ I testDecl YR value
    I HAS A TEST ITZ value
    FOUND YR TEST
IF U SAY SO

I HAS A TEST ITZ 3
VISIBLE "test is" TEST MKAY?,                   BTW TEST should be 3
VISIBLE "test is" testDecl 4 MKAY? MKAY?,       BTW TEST should be 4
VISIBLE "test is" TEST MKAY?,                   BTW TEST should be 3
VISIBLE "test is" testAssign 5 MKAY? MKAY?,     BTW TEST should be 5
VISIBLE "test is" TEST MKAY?,                   BTW TEST should be 5

KTHXBAI

Output:
test is 3
test is 4
test is 3
test is 5
test is 5
```

This test ensures that assigning a variable's value back to itself works correctly.

```
HAI
        I HAS A var1 ITZ 1
        I HAS A var2 ITZ 2.345
        I HAS A var3 ITZ "Lorem ipsum dolor sit"
        var1 R var1
        var2 R var2
        var3 R var3
        VISIBLE var1 var2 var3 MKAY?
KTHXBAI
```

Output:

```
1 2.345 Lorem ipsum dolor sit
```

This test checks to make sure that variable names are case sensitive.

```
HAI
        I HAS A var ITZ 1
        I HAS A Var ITZ 2.345
        I HAS A VAR ITZ "Lorem ipsum dolor sit"
        VISIBLE var Var VAR MKAY?
KTHXBAI
```

Output:

```
1 2.345 Lorem ipsum dolor sit
```

This test checks that variables may be initialized when declared.

```
HAI
        I HAS A var1 ITZ 1
        I HAS A var2 ITZ 2.345
        I HAS A var3 ITZ "Lorem ipsum dolor sit"
        VISIBLE var1 var2 var3 MKAY?
KTHXBAI
```

Output:

```
1 2.345 Lorem ipsum dolor sit
```

This test checks that variables may only be defined at most once.

```
HAI
        I HAS A var
        I HAS A var ITZ WIN
KTHXBAI
```

Output:

```
Error: Invalid declaration.
```

This test checks that an uninitialized variable starts out as none.

```
HAI
        I HAS A var
        VISIBLE var MKAY?
KTHXBAI
```

Output:

```
None
```

7-OPERATORS

1-Addition

This test checks that the addition operator allows the appropriate number of arguments.

```
HAI
        SUM OF 1 AN 3 AN 2
KTHXBAI
```

Output:

```
Error: invalid YARN.
```

This test makes sure the addition operator works correctly on floating point decimal and integer arguments.

```
HAI
        VISIBLE SUM OF 1.23 AN 4 MKAY?
KTHXBAI
```

Output:
```
5.23
```

This test makes sure the addition operator works correctly on floating point decimal arguments.

```
HAI
        VISIBLE SUM OF 1.23 AN 4.56 MKAY?
KTHXBAI
```

Output:

```
5.789999999999999
```

This test makes sure the addition operator works correctly on integer and floating point decimal arguments.

```
HAI
        VISIBLE SUM OF 1 AN 2.34 MKAY?
KTHXBAI
```

Output:
```
3.34
```

This test makes sure the addition operator works correctly on integer arguments.

```
HAI
        VISIBLE SUM OF 1 AN 2 MKAY?
KTHXBAI
```

Output:

```
3
```

This test checks that the addition operator can be nested as the argument of itself.

```
HAI
        VISIBLE SUM OF 1 AN SUM OF 3 AN 2 MKAY?
KTHXBAI
```

Output:

```
6
```

This test makes sure the addition operator returns an error on strings containing integer and/or float arguments.

```
HAI 1.2
        VISIBLE SUM OF "1" AN "2.75" MKAY?
KTHXBAI
```

Output:

```
12.75
```

7-OPERATORS

2-Subtraction

This test checks that the subtraction operator allows the appropriate number of arguments.

```
HAI
        DIFF OF 1 AN 3 AN 2
KTHXBAI
```

Output:

```
Error: invalid YARN.
```

This test makes sure the subtraction operator works correctly on floating point decimal and integer arguments.

```
HAI
        VISIBLE DIFF OF 1.23 AN 4 MKAY?
KTHXBAI
```

Output:

```
-2.77
```

This test makes sure the subtraction operator works correctly on floating point decimal arguments.

```
HAI
        VISIBLE DIFF OF 1.23 AN 4.56 MKAY?
KTHXBAI
```

Output:

```
-3.3299999999999996
```

This test makes sure the subtraction operator works correctly on integer and floating point decimal arguments.

```
HAI
        VISIBLE DIFF OF 2.34 AN 1 MKAY?
KTHXBAI
```

Output:

```
1.3399999999999999
```

This test makes sure the subtraction operator works correctly on integer arguments.

```
HAI
        VISIBLE DIFF OF 1 AN 2 MKAY?
KTHXBAI
```

Output:

```
-1
```

This test checks that the subtraction operator can be nested as the argument of itself.

```
HAI
        VISIBLE DIFF OF 1 AN SUM OF 3 AN 2 MKAY?
KTHXBAI
```

Output:

```
-4
```

This test makes sure the subtraction operator returns an error on strings containing integer and/or float arguments.

```
HAI 1.2
        VISIBLE DIFF OF "1" AN "2.75" MKAY?
KTHXBAI
```

Output:

```
Unexpected error: unsupported operand type(s) for -: 'str' and 'str'
```

7-OPERATORS
3-Multiplication

This test checks that the multiplication operator allows the appropriate number of arguments.

```
HAI
        PRODUKT OF 1 AN 3 AN 2
KTHXBAI
```

Output:
```
Error: invalid YARN.
```

This test makes sure the multiplication operator works correctly on floating point decimal and integer arguments.

```
HAI
        VISIBLE PRODUKT OF 1.23 AN 4 MKAY?
KTHXBAI
```

Output:

```
4.92
```

This test makes sure the multiplication operator works correctly on floating point decimal arguments.

```
HAI
        VISIBLE PRODUKT OF 1.23 AN 4.56 MKAY?
KTHXBAI
```

Output:

```
5.6088
```

This test makes sure the multiplication operator works correctly on integer and floating point decimal arguments.

```
HAI
        VISIBLE PRODUKT OF 1 AN 2.34 MKAY?
KTHXBAI
```

Output:

```
2.34
```

This test makes sure the multiplication operator works correctly on integer arguments.

```
HAI
        VISIBLE PRODUKT OF 1 AN 2 MKAY?
KTHXBAI
```

Output:

```
2
```

PRODUKTThis test checks that the multiplication operator can be nested as the argument of itself.

```
HAI
        VISIBLE PRODUKT OF 1 AN SUM OF 3 AN 2 MKAY?
KTHXBAI
```

Output:

```
5
```

This test makes sure the addition operator concatenates strings containing integer and/or float arguments.

```
HAI 1.2
        VISIBLE SUM OF "1" AN "2.75" MKAY?
KTHXBAI
```

Output:

```
12.75
```

7-OPERATORS

4-Division

This test checks that the division operator allows the appropriate number of arguments.

```
HAI
        QUOSHUNT OF 1 AN 3 AN 2
KTHXBAI
```

Output:
```
Error: invalid YARN.
```

This test makes sure that floating point decimal division by 0 raises an error.

```
HAI
        QUOSHUNT OF 1.0 AN 0.0
KTHXBAI
```

Output:

```
Unexpected error: float division by zero
```

This test makes sure the division operator works correctly on floating point decimal and integer arguments.

```
HAI
        VISIBLE QUOSHUNT OF 1.23 AN 4 MKAY?
KTHXBAI
```

Output:

```
0.3075
```

This test makes sure the division operator works correctly on floating point decimal arguments.

```
HAI
        VISIBLE QUOSHUNT OF 1.23 AN 4.56 MKAY?
KTHXBAI
```

Output:

```
0.26973684210526316
```

This test makes sure that integer division by 0 raises an error.

```
HAI
        QUOSHUNT OF 1 AN 0
KTHXBAI
```

Output:

```
Unexpected error: division by zero
```

This test makes sure the division operator works correctly on integer and floating point decimal arguments.

```
HAI
        VISIBLE QUOSHUNT OF 1 AN 2.34 MKAY?
KTHXBAI
```

Output:

```
0.4273504273504274
```

This test makes sure the division operator works correctly on integer arguments.

```
HAI
        VISIBLE QUOSHUNT OF 1 AN 2 MKAY?
KTHXBAI
```

Output:

```
0.5
```

This test checks that the division operator can be nested as the argument of itself.

```
HAI
        VISIBLE QUOSHUNT OF 1 AN SUM OF 3 AN 2 MKAY?
KTHXBAI
```

Output:

```
0.2
```

This test makes sure the division operator returns an error on strings containing integer and/or float arguments.

```
HAI 1.2
        VISIBLE QUOSHUNT OF "1" AN "2.75" MKAY?
KTHXBAI
```

Output:

```
Unexpected error: unsupported operand type(s) for /: 'str' and 'str'
```

7-OPERATORS

5-Modulo

This test checks that the modulo operator allows the appropriate number of arguments.

```
HAI
        MOD OF 1 AN 3 AN 2
KTHXBAI
```

Output:
```
Error: invalid YARN.
```

This test makes sure that floating point decimal modulo by 0 raises an error.

```
HAI
        MOD OF 1.0 AN 0.0
KTHXBAI
```

Output:
```
Unexpected error: float modulo
```

This test makes sure the modulo operator works correctly on floating point decimal and integer arguments.

```
HAI
        VISIBLE MOD OF 1.23 AN 4 MKAY?
KTHXBAI
```

Output:

```
1.23
```

This test makes sure the modulo operator works correctly on floating point decimal arguments.

```
HAI
        VISIBLE MOD OF 1.23 AN 4.56 MKAY?
KTHXBAI
```

Output:

```
1.23
```

This test makes sure that integer modulo by 0 raises an error.

```
HAI
        MOD OF 1 AN 0
KTHXBAI
```

Output:
```
Unexpected error: integer division or modulo by zero
```

This test makes sure the modulo operator works correctly on integer and floating point decimal arguments.

```
HAI
        VISIBLE MOD OF 1 AN 2.34 MKAY?
KTHXBAI
```

Output:

```
1.0
```

This test makes sure the modulo operator works correctly on integer arguments.

```
HAI
        VISIBLE MOD OF 1 AN 2 MKAY?
KTHXBAI
```

Output:

```
1
```

This test checks that the modulo operator can be nested as the argument of itself.

```
HAI
        VISIBLE MOD OF 1 AN SUM OF 3 AN 2 MKAY?
KTHXBAI
```

Output:

```
1
```

This test makes sure the modulo operator returns an error on strings containing integer and/or float arguments.

```
HAI 1.2
        VISIBLE MOD OF "1" AN "2.75" MKAY?
KTHXBAI
```

Output:

```
Unexpected error: not all arguments converted during string formatting
```

7-OPERATORS

6-Maximum

This test checks that the max operator allows the appropriate number of arguments.

```
HAI
        BIGGR OF 1 AN 3 AN 2
KTHXBAI
```

Output:
```
Error: invalid YARN.
```

This test makes sure the max operator works correctly on floating point decimal and integer arguments.

```
HAI
        VISIBLE BIGGR OF 1.23 AN 4 MKAY?
KTHXBAI
```

Output:
```
4
```

This test makes sure the max operator works correctly on floating point decimal arguments.

```
HAI
        VISIBLE BIGGR OF 1.23 AN 4.56 MKAY?
KTHXBAI
```

Output:

```
4.56
```

This test makes sure the max operator works correctly on integer and floating point decimal arguments.

```
HAI
        VISIBLE BIGGR OF 1 AN 2.34 MKAY?
KTHXBAI
```

Output:
```
2.34
```

This test makes sure the max operator works correctly on integer arguments.

```
HAI
        VISIBLE BIGGR OF 1 AN 2 MKAY?
KTHXBAI
```

Output:

```
2
```

This test checks that the max operator can be nested as the argument of itself.

```
HAI
        VISIBLE BIGGR OF 1 AN SUM OF 3 AN 2 MKAY?
KTHXBAI
```

Output:

```
5
```

This test makes sure the max operator works correctly on strings containing integer and/or float arguments.

```
HAI 1.2
        VISIBLE BIGGR OF "1" AN "2.75" MKAY?
KTHXBAI
```

Output:

```
2.75
```

7-OPERATORS

7-Minimum

This test checks that the min operator allows the appropriate number of arguments.

```
HAI
        SMALLR OF 1 AN 3 AN 2
KTHXBAI
```

Output:

```
Error: invalid YARN.
```

This test makes sure the min operator works correctly on floating point decimal and integer arguments.

```
HAI
        VISIBLE SMALLR OF 1.23 AN 4 MKAY?
KTHXBAI
```

Output:

```
1.23
```

This test makes sure the min operator works correctly on floating point decimal arguments.

```
HAI
        VISIBLE SMALLR OF 1.23 AN 4.56 MKAY?
KTHXBAI
```

Output:
```
1.23
```

This test makes sure the min operator works correctly on integer and floating point decimal arguments.

```
HAI
        VISIBLE SMALLR OF 1 AN 2.34 MKAY?
KTHXBAI
```

Output:

```
1
```

This test makes sure the min operator works correctly on integer arguments.

```
HAI
        VISIBLE SMALLR OF 1 AN 2 MKAY?
KTHXBAI
```

Output:

```
1
```

This test checks that the min operator can be nested as the argument of itself.

```
HAI
        VISIBLE SMALLR OF 1 AN SUM OF 3 AN 2 MKAY?
KTHXBAI
```

Output:

```
1
```

This test makes sure the min operator works correctly on strings containing integer and/or float arguments.

```
HAI 1.2
        VISIBLE SMALLR OF "1" AN "2.75" MKAY?
KTHXBAI
```

Output:

```
1
```

7-OPERATORS

8-Logical AND Binary

This test makes sure that the logical AND binary operator can be nested within itself. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        VISIBLE SUM OF 0 AN BOTH OF WIN AN BOTH OF WIN AN WIN MKAY?
        VISIBLE SUM OF 0 AN BOTH OF WIN AN BOTH OF WIN AN FAIL MKAY?
        VISIBLE SUM OF 0 AN BOTH OF FAIL AN BOTH OF WIN AN WIN MKAY?
        VISIBLE SUM OF 0 AN BOTH OF FAIL AN BOTH OF WIN AN FAIL MKAY?
KTHXBAI
```

Output:

```
1
0
0
0
```

This test checks that the logical AND binary operator produces the correct truth table. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        VISIBLE SUM OF 0 AN BOTH OF FAIL AN FAIL MKAY?
        VISIBLE SUM OF 0 AN BOTH OF FAIL AN WIN MKAY?
        VISIBLE SUM OF 0 AN BOTH OF WIN AN FAIL MKAY?
        VISIBLE SUM OF 0 AN BOTH OF WIN AN WIN MKAY?
KTHXBAI
```

Output:

```
0
0
0
1
```

7-OPERATORS

9-Logical OR Binary

This test makes sure that the logical OR binary operator can be nested within itself. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        VISIBLE SUM OF 0 AN EITHER OF WIN AN EITHER OF WIN AN WIN MKAY?
        VISIBLE SUM OF 0 AN EITHER OF WIN AN EITHER OF WIN AN FAIL MKAY?
        VISIBLE SUM OF 0 AN EITHER OF FAIL AN EITHER OF WIN AN WIN MKAY?
        VISIBLE SUM OF 0 AN EITHER OF FAIL AN EITHER OF WIN AN FAIL MKAY?
KTHXBAI
```

Output:

```
1
1
1
1
```

This test checks that the logical OR binary operator produces the correct truth table. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        VISIBLE SUM OF 0 AN EITHER OF FAIL AN FAIL MKAY?
        VISIBLE SUM OF 0 AN EITHER OF FAIL AN WIN MKAY?
        VISIBLE SUM OF 0 AN EITHER OF WIN AN FAIL MKAY?
        VISIBLE SUM OF 0 AN EITHER OF WIN AN WIN MKAY?
KTHXBAI
```

```
Output:
0
1
1
1
```

7-OPERATORS

10-Casting

This test checks that the explicit cast operator works correctly when casting to boolean types.

```
HAI 1.2
        I HAS A var
        I HAS A var2 ITZ 4
        I HAS A var3 ITZ 3.25
        I HAS A var4 ITZ "HAI MAUS"
        I HAS A var5 ITZ WIN
        VISIBLE SUM OF 0 AN NOT MAEK var A TROOF MKAY?
        VISIBLE SUM OF 0 AN NOT MAEK var2 A TROOF MKAY?
        VISIBLE SUM OF 0 AN NOT MAEK var3 A TROOF MKAY?
        VISIBLE SUM OF 0 AN NOT MAEK var4 A TROOF MKAY?
        VISIBLE SUM OF 0 AN NOT MAEK var5 A TROOF MKAY?
KTHXBAI
```

Output:

```
1
0
0
0
0
```

This test checks that the explicit cast operator works correctly when casting to floating point types.

```
HAI 1.2
        I HAS A var
        I HAS A var2 ITZ 4
        I HAS A var3 ITZ 3.25
        I HAS A var4 ITZ "HAI MAUS"
        I HAS A var5 ITZ WIN
        VISIBLE SUM OF 0 AN NOT MAEK var A NUMBAR MKAY?
        VISIBLE SUM OF 0 AN NOT MAEK var2 A NUMBAR MKAY?
        VISIBLE SUM OF 0 AN NOT MAEK var3 A NUMBAR MKAY?
        VISIBLE SUM OF 0 AN NOT MAEK var4 A NUMBAR MKAY?
        VISIBLE SUM OF 0 AN NOT MAEK var5 A NUMBAR MKAY?
KTHXBAI
```

Output:

```
Error: invalid cast.
```

This test checks that the explicit cast operator works correctly when casting to integer types.

```
HAI 1.2
        I HAS A var
        I HAS A var2 ITZ 4
        I HAS A var3 ITZ 3.25
        I HAS A var4 ITZ "HAI MAUS"
        I HAS A var5 ITZ WIN
        VISIBLE SUM OF 0 AN NOT MAEK var2 A NUMBR MKAY?
        VISIBLE SUM OF 0 AN NOT MAEK var3 A NUMBR MKAY?
        VISIBLE SUM OF 0 AN NOT MAEK var4 A NUMBR MKAY?
        VISIBLE SUM OF 0 AN NOT MAEK var5 A NUMBR MKAY?
        VISIBLE SUM OF 0 AN NOT MAEK var A NUMBR MKAY?
KTHXBAI
```

```
Output:
0
0
Unexpected error: invalid literal for int() with base 10: 'HAI MAUS'
```

This test checks that the explicit cast operator returns an error when casting to nil types.

```
HAI 1.2
        I HAS A var
        I HAS A var2 ITZ 4
        I HAS A var3 ITZ 3.25
        I HAS A var4 ITZ "HAI MAUS"
        I HAS A var5 ITZ WIN
        VISIBLE SUM OF 0 AN NOT MAEK var A NOOB MKAY?
        VISIBLE SUM OF 0 AN NOT MAEK var2 A NOOB MKAY?
        VISIBLE SUM OF 0 AN NOT MAEK var3 A NOOB MKAY?
        VISIBLE SUM OF 0 AN NOT MAEK var4 A NOOB MKAY?
        VISIBLE SUM OF 0 AN NOT MAEK var5 A NOOB MKAY?
KTHXBAI
```

Output:

```
Error: invalid cast.
```

7-OPERATORS

11-Logical NOT Unary

This test makes sure that the unary NOT operator can be nested within itself. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        VISIBLE SUM OF 0 AN EITHER OF WIN AN NOT NOT WIN MKAY?
        VISIBLE SUM OF 0 AN EITHER OF WIN AN NOT NOT FAIL MKAY?
        VISIBLE SUM OF 0 AN EITHER OF FAIL AN NOT NOT WIN MKAY?
        VISIBLE SUM OF 0 AN EITHER OF FAIL AN NOT NOT FAIL MKAY?
KTHXBAI
```

Output:

```
1
1
1
0
```

This test checks that the unary NOT operator produces the correct truth table. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        VISIBLE SUM OF 0 AN NOT FAIL MKAY?
        VISIBLE SUM OF 0 AN NOT WIN MKAY?
        VISIBLE SUM OF 0 AN NOT FAIL MKAY?
        VISIBLE SUM OF 0 AN NOT WIN MKAY?
KTHXBAI
```

Output:

```
1
0
1
0
```

7-OPERATORS

12-Logical AND N-ary

This test checks that the logical AND n-ary operator works correctly with many arguments. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI 1.2
        VISIBLE SUM OF 0 AN ALL OF WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN
WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN
WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN
WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN
WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN MKAY?
MKAY?
        VISIBLE SUM OF 0 AN ALL OF WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN
WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN
WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN
WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN
WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN FAIL
MKAY? MKAY?
KTHXBAI
```

Output:
```
1
0
```

This test makes sure that the logical AND n-ary operator can be nested within itself. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        VISIBLE SUM OF 0 AN ALL OF WIN AN ALL OF WIN AN WIN AN WIN MKAY? AN
WIN MKAY? MKAY?
        VISIBLE SUM OF 0 AN ALL OF WIN AN ALL OF WIN AN WIN AN FAIL MKAY?
AN WIN MKAY? MKAY?
        VISIBLE SUM OF 0 AN ALL OF WIN AN ALL OF WIN AN WIN AN WIN MKAY? AN
FAIL MKAY? MKAY?
        VISIBLE SUM OF 0 AN ALL OF WIN AN ALL OF WIN AN WIN AN FAIL MKAY?
AN FAIL MKAY? MKAY?
KTHXBAI
```

Output:

```
1
0
0
0
```

This test checks that the logical AND n-ary operator produces the correct truth table. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        VISIBLE SUM OF 0 AN ALL OF FAIL AN FAIL AN FAIL AN FAIL MKAY? MKAY?
        VISIBLE SUM OF 0 AN ALL OF FAIL AN FAIL AN FAIL AN WIN MKAY? MKAY?
        VISIBLE SUM OF 0 AN ALL OF FAIL AN FAIL AN WIN AN FAIL MKAY? MKAY?
        VISIBLE SUM OF 0 AN ALL OF FAIL AN FAIL AN WIN AN WIN MKAY? MKAY?
        VISIBLE SUM OF 0 AN ALL OF FAIL AN WIN AN FAIL AN FAIL MKAY? MKAY?
        VISIBLE SUM OF 0 AN ALL OF FAIL AN WIN AN FAIL AN WIN MKAY? MKAY?
        VISIBLE SUM OF 0 AN ALL OF FAIL AN WIN AN WIN AN FAIL MKAY? MKAY?
        VISIBLE SUM OF 0 AN ALL OF FAIL AN WIN AN WIN AN WIN MKAY? MKAY?
        VISIBLE SUM OF 0 AN ALL OF WIN AN FAIL AN FAIL AN FAIL MKAY? MKAY?
        VISIBLE SUM OF 0 AN ALL OF WIN AN FAIL AN FAIL AN WIN MKAY? MKAY?
        VISIBLE SUM OF 0 AN ALL OF WIN AN FAIL AN WIN AN FAIL MKAY? MKAY?
        VISIBLE SUM OF 0 AN ALL OF WIN AN FAIL AN WIN AN WIN MKAY? MKAY?
        VISIBLE SUM OF 0 AN ALL OF WIN AN WIN AN FAIL AN FAIL MKAY? MKAY?
        VISIBLE SUM OF 0 AN ALL OF WIN AN WIN AN FAIL AN WIN MKAY? MKAY?
        VISIBLE SUM OF 0 AN ALL OF WIN AN WIN AN WIN AN FAIL MKAY? MKAY?
        VISIBLE SUM OF 0 AN ALL OF WIN AN WIN AN WIN AN WIN MKAY? MKAY?
KTHXBAI
```

Output:

```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
```

7-OPERATORS

13-Logical OR N-ary

This test checks that the logical OR n-ary operator works correctly with many arguments. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI 1.2
        VISIBLE SUM OF 0 AN ANY OF WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN
WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN
WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN
WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN
WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN MKAY?
MKAY?
        VISIBLE SUM OF 0 AN ANY OF WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN
WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN
WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN
WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN
WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN
AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN WIN AN FAIL
MKAY? MKAY?
KTHXBAI
```

Output:
1
1

This test makes sure that the logical OR n-ary operator can be nested within itself. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        VISIBLE SUM OF 0 AN ANY OF WIN AN ANY OF WIN AN WIN AN WIN MKAY? AN
WIN MKAY? MKAY?
        VISIBLE SUM OF 0 AN ANY OF WIN AN ANY OF WIN AN WIN AN FAIL MKAY?
AN WIN MKAY? MKAY?
        VISIBLE SUM OF 0 AN ANY OF WIN AN ANY OF WIN AN WIN AN WIN MKAY? AN
FAIL MKAY? MKAY?
        VISIBLE SUM OF 0 AN ANY OF WIN AN ANY OF WIN AN WIN AN FAIL MKAY?
AN FAIL MKAY? MKAY?
KTHXBAI
```

Output:

```
1
1
1
1
```

This test checks that the logical OR n-ary operator produces the correct truth table. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        VISIBLE SUM OF 0 AN ANY OF FAIL AN FAIL AN FAIL AN FAIL MKAY? MKAY?
        VISIBLE SUM OF 0 AN ANY OF FAIL AN FAIL AN FAIL AN WIN MKAY? MKAY?
        VISIBLE SUM OF 0 AN ANY OF FAIL AN FAIL AN WIN AN FAIL MKAY? MKAY?
        VISIBLE SUM OF 0 AN ANY OF FAIL AN FAIL AN WIN AN WIN MKAY? MKAY?
        VISIBLE SUM OF 0 AN ANY OF FAIL AN WIN AN FAIL AN FAIL MKAY? MKAY?
        VISIBLE SUM OF 0 AN ANY OF FAIL AN WIN AN FAIL AN WIN MKAY? MKAY?
        VISIBLE SUM OF 0 AN ANY OF FAIL AN WIN AN WIN AN FAIL MKAY? MKAY?
        VISIBLE SUM OF 0 AN ANY OF FAIL AN WIN AN WIN AN WIN MKAY? MKAY?
        VISIBLE SUM OF 0 AN ANY OF WIN AN FAIL AN FAIL AN FAIL MKAY? MKAY?
        VISIBLE SUM OF 0 AN ANY OF WIN AN FAIL AN FAIL AN WIN MKAY? MKAY?
        VISIBLE SUM OF 0 AN ANY OF WIN AN FAIL AN WIN AN FAIL MKAY? MKAY?
        VISIBLE SUM OF 0 AN ANY OF WIN AN FAIL AN WIN AN WIN MKAY? MKAY?
        VISIBLE SUM OF 0 AN ANY OF WIN AN WIN AN FAIL AN FAIL MKAY? MKAY?
        VISIBLE SUM OF 0 AN ANY OF WIN AN WIN AN FAIL AN WIN MKAY? MKAY?
        VISIBLE SUM OF 0 AN ANY OF WIN AN WIN AN WIN AN FAIL MKAY? MKAY?
        VISIBLE SUM OF 0 AN ANY OF WIN AN WIN AN WIN AN WIN MKAY? MKAY?
KTHXBAI
```

Output:

```
0
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

7-OPERATORS

14-Equality

This test checks that the equality operator works correctly with boolean typed arguments. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        I HAS A var1 ITZ WIN
        I HAS A var2 ITZ FAIL
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN FAIL MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM FAIL AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN WIN MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM WIN AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 0 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 1 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 0.0 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 0.0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 1.234 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 1.234 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "0" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "1" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "1" AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "0.0" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "0.0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "1.234" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "1.234" AN var2 MKAY?
KTHXBAI
```

```
Output:
0
0
1
1
0
0
1
1
0
0
1
0
0
0
```

0
0
0
0
0
0
0

This test checks that the equality operator works correctly with floating point decimal typed arguments. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        I HAS A var1 ITZ 0.0
        I HAS A var2 ITZ 1.234
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN FAIL MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM FAIL AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN WIN MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM WIN AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 0 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 1 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 0.0 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 0.0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 1.234 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 1.234 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "0" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "1" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "1" AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "0.0" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "0.0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "1.234" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "1.234" AN var2 MKAY?
KTHXBAI
```

Output:

```
0
1
0
0
0
1
0
0
0
1
0
0
1
0
```

0
0
0
0
0
0
0

This test checks that the equality operator works correctly with integer typed arguments. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        I HAS A var1 ITZ 0
        I HAS A var2 ITZ 1
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN FAIL MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM FAIL AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN WIN MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM WIN AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 0 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 1 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 0.0 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 0.0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 1.234 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 1.234 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "0" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "1.234" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "1.234" AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "0.0" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "0.0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "1.234" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "1.234" AN var2 MKAY?
KTHXBAI
```

```
Output:
0
1
0
0
1
1
0
0
1
1
0
0
0
0
```

0
0
0
0
0
0
0

This test checks that the equality operator works correctly with nil typed arguments. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        I HAS A var1
        I HAS A var2
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN FAIL MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM FAIL AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN WIN MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM WIN AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 0 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 1 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 0.0 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 0.0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 1.234 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 1.234 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "0" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "1" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "1" AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "0.0" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "0.0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "1.234" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "1.234" AN var2 MKAY?
KTHXBAI
```

Output:

```
1
0
0
0
0
0
0
0
0
0
0
0
0
0
```

0
0
0
0
0
0
0

This test checks that the equality operator works correctly with string typed arguments. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        I HAS A var1 ITZ "0"
        I HAS A var2 ITZ "1.234"
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN FAIL MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM FAIL AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN WIN MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM WIN AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 0 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 1 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 0.0 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 0.0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN 1.234 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM 1.234 AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "0" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "1" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "1" AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "0.0" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "0.0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM var1 AN "1.234" MKAY?
        VISIBLE SUM OF 0 AN BOTH SAEM "1.234" AN var2 MKAY?
KTHXBAI
```

```
Output:
0
0
0
0
0
0
0
0
0
0
0
0
0
1
```

0
0
0
0
0
0
1

7-OPERATORS

15-Inequality

This test checks that the inequality operator works correctly with boolean typed arguments. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        I HAS A var1 ITZ WIN
        I HAS A var2 ITZ FAIL
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN FAIL MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT FAIL AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN WIN MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT WIN AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 0 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 1 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 0.0 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 0.0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 1.234 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 1.234 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "0" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "1" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "1" AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "0.0" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "0.0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "1.234" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "1.234" AN var2 MKAY?
KTHXBAI
```

Output:

```
1
1
0
0
1
1
0
0
1
1
0
1
1
1
```

1
1
1
1
1
1
1

This test checks that the inequality operator works correctly with floating point decimal typed arguments. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        I HAS A var1 ITZ 0.0
        I HAS A var2 ITZ 1.234
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN FAIL MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT FAIL AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN WIN MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT WIN AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 0 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 1 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 0.0 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 0.0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 1.234 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 1.234 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "0" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "1" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "1" AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "0.0" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "0.0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "1.234" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "1.234" AN var2 MKAY?
KTHXBAI
```

Output:

```
1
0
1
1
1
0
1
1
1
0
1
1
0
1
```

1
1
1
1
1
1
1

This test checks that the inequality operator works correctly with integer typed arguments. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        I HAS A var1 ITZ 0
        I HAS A var2 ITZ 1
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN FAIL MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT FAIL AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN WIN MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT WIN AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 0 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 1 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 0.0 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 0.0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 1.234 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 1.234 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "0" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "1.234" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "1.234" AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "0.0" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "0.0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "1.234" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "1.234" AN var2 MKAY?
KTHXBAI
```

Output:

```
1
0
1
1
0
0
1
1
0
0
1
1
1
1
```

1
1
1
1
1
1
1

This test checks that the inequality operator works correctly with nil typed arguments. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        I HAS A var1
        I HAS A var2
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN FAIL MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT FAIL AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN WIN MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT WIN AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 0 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 1 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 0.0 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 0.0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 1.234 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 1.234 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "0" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "1" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "1" AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "0.0" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "0.0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "1.234" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "1.234" AN var2 MKAY?
KTHXBAI
```

Output:

```
0
1
1
1
1
1
1
1
1
1
1
1
1
```

1
1
1
1
1
1
1

This test checks that the inequality operator works correctly with string typed arguments. Note that boolean values can not be cast to strings so this test first implicitly casts the resulting boolean values to integer values and prints these out instead.

```
HAI
        I HAS A var1 ITZ "0"
        I HAS A var2 ITZ "1.234"
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN FAIL MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT FAIL AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN WIN MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT WIN AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 0 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 1 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 1 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 0.0 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 0.0 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN 1.234 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT 1.234 AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "0" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "1" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "1" AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "0.0" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "0.0" AN var2 MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT var1 AN "1.234" MKAY?
        VISIBLE SUM OF 0 AN DIFFRINT "1.234" AN var2 MKAY?
KTHXBAI
```

Output:

```
1
1
1
1
1
1
1
1
1
1
1
1
1
0
```

1
1
1
1
1
1
0

8-CONDITIONALS

This test checks that the operation of the else branch in an if/then/else block works correctly.

```
HAI
        I HAS A var
        var
        O RLY?
                YA RLY, VISIBLE "1a" MKAY?
                NO WAI, VISIBLE "1b" MKAY?
        OIC
        FAIL
        O RLY?
                YA RLY, VISIBLE "2a" MKAY?
                NO WAI, VISIBLE "2b" MKAY?
        OIC
        0
        O RLY?
                YA RLY
                        VISIBLE "3a" MKAY?
                NO WAI
                        VISIBLE "3b" MKAY?
        OIC
        0.0
        O RLY?
                YA RLY
                        VISIBLE "4a" MKAY?
                NO WAI
                        VISIBLE "4b" MKAY?
        OIC
        ""
        O RLY?
                YA RLY
                        VISIBLE "5a" MKAY?
                NO WAI
                        VISIBLE "5b" MKAY?
        OIC
KTHXBAI
```

Output:

```
1b
2b
3b
4b
5b
```

This test checks that the operation of the else-if branch in an if/then/ else block works correctly.

```
HAI
        FAIL
        O RLY?
                YA RLY
                        VISIBLE "1a" MKAY?
                MEBBE BOTH OF WIN AN WIN
                        VISIBLE "1b" MKAY?
        OIC
        FAIL
        O RLY?
                YA RLY
                        VISIBLE "2a" MKAY?
                MEBBE BOTH OF WIN AN WIN
                        VISIBLE "2b" MKAY?
                NO WAI
                        VISIBLE "2c" MKAY?
        OIC
        FAIL
        O RLY?
                YA RLY
                        VISIBLE "3a" MKAY?
                MEBBE BOTH OF WIN AN FAIL
                        VISIBLE "3b" MKAY?
                NO WAI
                        VISIBLE "3c" MKAY?
        OIC
        FAIL
        O RLY?
                YA RLY
                        VISIBLE "4a" MKAY?
                MEBBE BOTH OF WIN AN FAIL
                        VISIBLE "4b" MKAY?
        OIC
        FAIL
        O RLY?
                YA RLY
                        VISIBLE "5a" MKAY?
                MEBBE EITHER OF FAIL AN FAIL
                        VISIBLE "5b" MKAY?
                MEBBE BOTH OF WIN AN FAIL
                        VISIBLE "5c" MKAY?
                MEBBE ANY OF FAIL AN FAIL AN FAIL AN FAIL AN WIN MKAY?
                        VISIBLE "5d" MKAY?
```

```
                NO WAI
                        VISIBLE "5e" MKAY?
        OIC
KTHXBAI
```

Output:

```
1b
2b
3c
5d
```

This test checks that the operation of the if branch in a simple if/then/ else block works correctly.

```
HAI
        WIN
        O RLY?
                YA RLY
                        VISIBLE "1a" MKAY?
        OIC
        1
        O RLY?
                YA RLY
                        VISIBLE "2a" MKAY?
        OIC
        2.345
        O RLY?
                YA RLY
                        VISIBLE "3a" MKAY?
        OIC
        "abc"
        O RLY?
                YA RLY
                        VISIBLE "4a" MKAY?
        OIC
KTHXBAI
```

Output:

```
1a
2a
3a
4a
```

9-FUNCTIONS

This test checks that a set of functions can call themselves.

```
HAI
        HOW DUZ I fun1 YR a
                VISIBLE a MKAY?
                BOTH SAEM a AN 0
                O RLY?
                        YA RLY
                                FOUND YR NOOB
                OIC
                fun2 DIFF OF a AN 1 MKAY?
        IF U SAY SO

        HOW DUZ I fun2 YR a
                VISIBLE a MKAY?
                BOTH SAEM a AN 0
                O RLY?
                        YA RLY
                                FOUND YR NOOB
                OIC
                fun1 DIFF OF a AN 1 MKAY?
        IF U SAY SO

        fun1 9 MKAY?
KTHXBAI
```

Output:

```
9
8
7
6
5
4
3
2
1
0
```

This test checks that a function with an empty body works correctly.

```
HAI
        HOW DUZ I fun
        IF U SAY SO

        fun MKAY?
KTHXBAI
```

Output:

This test checks that *a* function with expressions for arguments works correctly.

```
HAI 1.2
        HOW DUZ I fun YR a
                VISIBLE a MKAY?
        IF U SAY SO

        fun SUM OF 1 AN 2.345 MKAY?
KTHXBAI
```

Output:

```
3.345
```

```
HAI,            BTW Fibonacci Numbers

HOW DUZ I FIB YR MAX
    I HAS A NUM ITZ 1
    I HAS A NEXT ITZ 1
    I HAS A COUNT ITZ 0

    DIFFRINT COUNT AN BIGGR OF COUNT AN MAX,    BTW if COUNT < MAX
    O RLY?
        YA RLY
            VISIBLE NEXT MKAY?
            COUNT R SUM OF COUNT AN 1,          BTW COUNT++
    OIC

    DIFFRINT COUNT AN BIGGR OF COUNT AN MAX,    BTW if COUNT < MAX
    O RLY?
        YA RLY
            VISIBLE NEXT MKAY?
            COUNT R SUM OF COUNT AN 1,          BTW COUNT++
            NEXT R SUM OF NEXT AN NUM,          BTW NEXT = NEXT + NUM
    OIC

    BTW for(; COUNT < MAX; COUNT++)
    IM IN YR loop WILE DIFFRINT COUNT AN BIGGR OF COUNT AN MAX
        VISIBLE NEXT MKAY?
        SUM OF NEXT AN NUM,             BTW IT = NEXT + NUM
        NUM R NEXT,                     BTW NUM = NEXT
        NEXT R IT,                      BTW NEXT = IT
        COUNT R SUM OF COUNT AN 1,      BTW COUNT += 1
    IM OUTTA YR loop

IF U SAY SO

I HAS A MAX

VISIBLE "How many fibonacci numbers do you wish to see?" MKAY?
GIMMEH MAX

MAX R MAEK MAX A NUMBR

FIB MAX MKAY?

KTHXBAI
```

Output:

```
How many fibonacci numbers do you wish to see?
```

```
LOL>> 20
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
```

This test checks that a function with a nil return value works correctly. Note that nil values cannot be implicitly cast to string values directly and thus this test first implicitly casts them to boolean values which are then cast to integer values which are then cast to string values.

```
HAI 1.2
        HOW DUZ I fun1
                "a"
                FOUND YR NOOB
        IF U SAY SO

        HOW DUZ I fun2 YR arg
                arg
                FOUND YR NOOB
        IF U SAY SO

        VISIBLE SUM OF 0 AN NOT fun1 MKAY? MKAY?
        VISIBLE SUM OF 0 AN NOT fun2 "b" MKAY? MKAY?
KTHXBAI
```

Output:

```
1
1
```

```
Recursively calculates the given base to the given power.
HAI,    BTW power function

HOW DUZ I pow YR base AN YR exp
        BTW Base Cases
        BOTH SAEM exp AN 0, O RLY?
                YA RLY, FOUND YR 1
        OIC

        BOTH SAEM exp AN 1, O RLY?
                YA RLY, FOUND YR base
        OIC

        BTW Recursive case
        I HAS A num
        I HAS A newExp

        QUOSHUNT OF exp AN 2,           BTW IT = exp / 2
        newExp R MAEK IT A NUMBR,       BTW newExp = floor(IT)

        OBTW Checking to see if the exponent is odd.
        If it is then we set IT = base, otherwise IT = 1 TLDR
        DIFFRINT 0 AN MOD OF exp AN 2, O RLY?
                YA RLY, IT R base
                NO WAI, IT R 1
        OIC

        num R pow base newExp MKAY?,    BTW num = pow(base exp)
        num R PRODUKT OF num AN num,    BTW num = num * num

        FOUND YR PRODUKT OF IT AN num,  BTW return IT * num
IF U SAY SO

I HAS A base
I HAS A exp

VISIBLE "What is the base?" MKAY?
GIMMEH base

BTW base R MAEK base A NUMBR

VISIBLE "What is the exponent?" MKAY?
GIMMEH exp

BTW exp R MAEK exp A NUMBR
```

```
VISIBLE base "to the" exp "is" pow base exp MKAY? MKAY?

KTHXBAI
```

Output:

```
What is the base?
LOL>> 4
What is the exponent?
LOL>> 32
4 to the 32 is 18446744073709551616
```

This test checks that a function can call itself.

```
HAI
        HOW DUZ I fun YR a
                VISIBLE a MKAY?
                BOTH SAEM a AN 0
                O RLY?
                        YA RLY
                                FOUND YR NOOB
                OIC
                fun DIFF OF a AN 1 MKAY?
        IF U SAY SO

        fun 9 MKAY?
KTHXBAI
```

Output:

```
9
8
7
6
5
4
3
2
1
0
```

```
HAI,    BTW testing return statements

HOW DUZ I testReturn YR code
    BOTH SAEM code AN 1
    O RLY?
        YA RLY
            FOUND YR 10
        MEBBE BOTH SAEM code AN 2
            FOUND YR 20
        MEBBE BOTH SAEM code AN 3
            FOUND YR 30
        NO WAI
            FOUND YR NOOB
    OIC
IF U SAY SO

VISIBLE testReturn 1 MKAY? MKAY?
VISIBLE testReturn 2 MKAY? MKAY?
VISIBLE testReturn 3 MKAY? MKAY?
VISIBLE testReturn NOOB MKAY? MKAY?

KTHXBAI
```

Output:

```
10
20
30
None
```

```
HAI,    BTW testing return statements

HOW DUZ I testReturn YR code
    BOTH SAEM code AN 1
    O RLY?
        YA RLY
            FOUND YR 10
        MEBBE BOTH SAEM code AN 2
            FOUND YR 20
        MEBBE BOTH SAEM code AN 3
            FOUND YR 30
        NO WAI
            FOUND YR NOOB
    OIC
IF U SAY SO

VISIBLE testReturn 1 MKAY? MKAY?
VISIBLE testReturn 2 MKAY? MKAY?
VISIBLE testReturn 3 MKAY? MKAY?
VISIBLE testReturn NOOB MKAY? MKAY?

FOUND YR NOOB

KTHXBAI
```

```
Output:
10
20
30
None
Unexpected error: Found return statement outside function.
```

This test checks that a function with a return value works correctly.

```
HAI
        HOW DUZ I fun1
                FOUND YR "a"
        IF U SAY SO

        HOW DUZ I fun2 YR arg
                FOUND YR arg
        IF U SAY SO

        VISIBLE fun1 MKAY? MKAY?
        VISIBLE fun2 "b" MKAY? MKAY?
KTHXBAI
```

Output:

```
a
b
```

This test checks that passing too many arguments to a function is not allowed.

```
HAI 1.2
        HOW DUZ I fun YR a
                VISIBLE a MKAY?
        IF U SAY SO

        fun MKAY?
KTHXBAI
```

Output:
```
Error: invalid number of arguments in function call.
```

This test checks that passing too many arguments to a function is not allowed.

```
HAI 1.2
        HOW DUZ I fun YR a
                VISIBLE a MKAY?
        IF U SAY SO

        fun 1 2 MKAY?
KTHXBAI
```

Output:

```
Error: invalid number of arguments in function call.
```

This test checks that a function with no explicit return value and side effects works correctly.

```
HAI
        HOW DUZ I fun1
                VISIBLE "a" MKAY?
        IF U SAY SO

        HOW DUZ I fun2 YR arg
                VISIBLE arg MKAY?
        IF U SAY SO

        fun1 MKAY?
        fun2 "b" MKAY?
KTHXBAI
```

Output:

```
a
b
```

10-LOOPS

```
HAI
I HAS A VAR ITZ 0

BTW for(; VAR <= 10; VAR++)
IM IN YR LOOP WILE BOTH SAEM VAR AN SMALLR OF VAR AN 10
        VISIBLE VAR MKAY?
        VAR R SUM OF VAR AN 1
IM OUTTA YR LOOP
KTHXBAI
```

Output:

```
0
1
2
3
4
5
6
7
8
9
10
```

```
HAI,            BTW Fibonacci Numbers
I HAS A MAX
I HAS A NUM ITZ 1
I HAS A NEXT ITZ 1
I HAS A COUNT ITZ 0

VISIBLE "How many fibonacci numbers do you wish to see?" MKAY?
GIMMEH MAX

MAX R MAEK MAX A NUMBR

DIFFRINT COUNT AN BIGGR OF COUNT AN MAX,        BTW if COUNT < MAX
O RLY?
    YA RLY
        VISIBLE NEXT MKAY?
        COUNT R SUM OF COUNT AN 1,              BTW COUNT++
OIC

DIFFRINT COUNT AN BIGGR OF COUNT AN MAX,        BTW if COUNT < MAX
O RLY?
    YA RLY
        VISIBLE NEXT MKAY?
        COUNT R SUM OF COUNT AN 1,              BTW COUNT++
        NEXT R SUM OF NEXT AN NUM,              BTW NEXT = NEXT + NUM
OIC

BTW for(; COUNT < MAX; COUNT++)
IM IN YR loop WILE DIFFRINT COUNT AN BIGGR OF COUNT AN MAX
    VISIBLE NEXT MKAY?
    SUM OF NEXT AN NUM,         BTW IT = NEXT + NUM
    NUM R NEXT,                 BTW NUM = NEXT
    NEXT R IT,                  BTW NEXT = IT
    COUNT R SUM OF COUNT AN 1,  BTW COUNT += 1
IM OUTTA YR loop

KTHXBAI
```

Output:

```
How many fibonacci numbers do you wish to see?
LOL>> 15
1
1
2
3
5
8
```

13
21
34
55
89
144
233
377
610

This test checks that loops with a "while" ending condition work correctly.

```
HAI
        I HAS A var ITZ 0
        IM IN YR loop WILE DIFFRINT var AN -10
                VISIBLE var MKAY?
                var R DIFF OF var AN 1
        IM OUTTA YR loop
KTHXBAI
```

Output:

```
0
-1
-2
-3
-4
-5
-6
-7
-8
-9
```